FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Sistema Multiplataforma de Proteção de Documentos Médicos por via de Assinaturas Digitais

Fernando Guilherme Ferreira Sousa

Mestrado Integrado em Engenharia Informática e Computação

Orientador: José Manuel de Magalhães Cruz (Prof. Doutor)

Julho de 2014

Sistema Multiplataforma de Proteção de Documentos Médicos por via de Assinaturas Digitais

Fernando Guilherme Ferreira Sousa

Mestrado Integrado em Engenharia Informática e Computação

Aprovado em provas públicas pelo Júri:

Presidente: Daniel Augusto Gama de Castro Silva (Prof. Doutor)

Vogal Externo: Marco Paulo Amorim Vieira (Prof. Doutor)

Orientador: José Manuel de Magalhães Cruz (Prof. Doutor)

---

14 de Julho de 2014

Resumo

O Sistema Multiplataforma de Proteção de Documentos Médicos por via de Assinaturas Digitais foca-se na proteção da integridade de um documento médico. Pretende-se garantir que um documento elaborado por um médico não seja alterado sem deteção, garantindo assim a confiança máxima na autoria do documento. Para facilitar a utilização do sistema desenvolvido a *interface* é multiplataforma, em termos de dispositivo eletrónico e sistema operativo. Um profissional médico, no final de elaborar um documento, assina-o digitalmente, com a sua chave secreta que irá estar contida e protegida dentro de um *Smart Card*. Depois, o documento é enviado a um sistema central que o valida e guarda numa base de dados. Mais tarde e sempre que o documento for acedido, será verificado e validado para evitar fraudes.

Num mundo cada vez mais digitalizado e com cada vez maior necessidade de garantir que todos os procedimentos na área da saúde são fidedignos, é necessário que qualquer documento que um médico assine seja considerado íntegro e que possa ser detetada qualquer forma de adulteração.

O sistema desenvolvido é um contributo nesse sentido.

Abstract

The Cross-Platform System of Medical Documents Protection through Digital Signatures focuses on protecting the integrity of a medical document. It must be ensured that computer security is involved, so that a document made by a doctor is not modified, ensuring with maximum confidence the document's authorship. To facilitate the use of the developed system, all the entire interface is a cross-platform one, for both electronic device and operative systems. A professional doctor, at the end of drafting a document, signs it digitally, by giving his secret key that may be contained, for instance, inside a smart card. Afterwards, the document is sent to a central system that validates and stores it in a database. Posteriorly, each time the document is accessed, it will be verified and validated, in order to avoid frauds.

In a world more and more digitalized, with necessities of ensuring that every procedures, in health area, are trusted, it is necessary that every document signed by a doctor be considered integral and valid.

The developed system is a contribution towards that.

Agradecimentos

Ao longo deste desenvolvimento da dissertação recebi diversos apoios e motivações de várias pessoas a quem estou imensamente grato.

Desde já faço um agradecimento especial ao meu orientador, Professor Doutor José Magalhães Cruz que não hesitou, desde logo, em aceitar o meu convite para me orientar, e também por toda a disponibilidade demonstrada, e ainda pelo facto de me guiar nas principais etapas da realização da dissertação.

À empresa Glintt Healthcare Solutions, e em particular ao Engenheiro Pedro Rocha, que me escolheu e confiou em mim para trabalhar com eles neste projeto, tendo sido também uma grande ajuda na discussão das diversas fases do projeto.

Aos meus pais, por todo o apoio, estabilidade, amor e disponibilidade incondicionais.

À Filipa, pela força, amor e carinho enormes que me transmitiu.

À Paula, que foi uma verdadeira amiga e peça fundamental para que eu pudesse manter-me focado na realização desta dissertação.

Fernando Guilherme Ferreira Sousa

Conteúdo

Lista de Figuras

Lista de Tabelas

Abreviaturas e Símbolos

AES	Advanced Encryption Standard
APDU	Application Protocol Data Unit
API	Application Programming Interface
CA	Certification Authority
CMP	Certification Management Protocol
COM	Component Object Model
CRL	Certification Revocation List
DLL	Dynamic-Link Library
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
IDE	Integrated Development Environment
IP	Internet Protocol
KRA	Key Recovery Agent
KRS	Key Recovery System
MHA	Mobile Home Agent
MVC	Model-View-Controller
PDF	Portable Document Format
PIN	Personal Identification Number
PKCS	Public-Key Cryptography Standards
PKI	Public-Key Infrastructure
POP	Proof of Possession
SSL	Secure Sockets Layer
TLS	Transport Layer Security
TTP	Trusted Third Party
URL	Uniform Resource Locator
WWW	World Wide Web
XML	Extensible Markup Language

Capítulo 1

1. Introdução

Hoje em dia, em transações médicas, é cada vez mais necessária uma digitalização de todos os procedimentos que os médicos efetuam, desde consultas que marcam às receitas que passam aos seus pacientes. É, por isso, fundamental garantir a integridade dos procedimentos e impedir fraudes.

Uma forma de conseguir tal integridade consiste em fazer com que todas as transações que só dizem respeito a um médico sejam assinadas digitalmente por ele, através de um *Smart Card* pessoal e intransmissível, que contenha a sua chave criptográfica privada.

Para que o médico não tenha de estar obrigatoriamente ambientado a um certo tipo de sistema operativo ou plataforma, tanto num dispositivo fixo como móvel, é necessário criar um Sistema Multiplataforma que opere e possa ser utilizado pelos sistemas operativos mais comuns. Desta forma a universalidade do sistema fica garantida, podendo ser utilizada pelos mais variados profissionais da saúde, dando-lhes a oportunidade de utilizarem a plataforma em que se sentem mais confortáveis. Será este sistema que irá efetuar e validar a assinatura digital do médico de forma a verificar a sua integridade, recorrendo para isso a uma infraestrutura do tipo *Public-Key*. Para isso, após a elaboração de um documento, o médico irá inserir o seu *Smart Card* num leitor de cartões que está ligado ao dispositivo informático que ele estiver a utilizar e, antes de assinar, é efetuada uma validação para garantir com confiança a autenticidade do médico em questão, através de uma senha, por exemplo. Esta senha evita, assim, que outro sujeito se faça passar pelo médico, apenas por ter o seu cartão (afinal um dos princípios básicos de segurança passa por algo que só alguém tem juntamente com algo que só esse alguém sabe). Só após a inserção correta da senha é que a assinatura digital pode ser feita.

1.1 Enquadramento

Este tema insere-se sobretudo na área da saúde, porém poderá ser generalizado a outras áreas em que as características de funcionamento sejam semelhantes ao problema em causa. Tem havido cada vez mais uma grande necessidade de se digitalizar todos os processos na sociedade em geral e as organizações médicas não fogem a essa mesma necessidade, para garantirem maior controlo e organização. Está-se a tornar cada vez mais urgente a necessidade de gerir todos os processos individuais de saúde em formato digital, havendo então uma necessidade de assegurar segurança a todo este processo. Para agilizar o trabalho dos médicos é necessário que estes possam rapidamente elaborar documentos médicos e com a digitalização de tudo isto, e tendo em conta a extrema sensibilidade da área da saúde, é necessário que haja a certeza que não há qualquer tipo de fraude e ilegalidade envolvida na transação dos documentos entre as diversas entidades médicas, como os hospitais, farmácias, etc. As necessidades de segurança, em praticamente todos os tipos de aplicações informáticas, estão a crescer, porque se requer maior fiabilidade e maior confiança em todos os dados, incluindo na sua origem. Na área da saúde este cuidado deve ser reforçado, pois mais ninguém para além do médico deve poder elaborar certos documentos. É preciso garantir que há a certeza de que um documento digital elaborado por um determinado médico, foi mesmo elaborado por ele.

Esta dissertação foi proposta pela empresa Glintt - Healthcare Solutions, S.A. e nela desenvolvida. Naturalmente, todos os cenários, sejam de implementação ou de teste, foram produzidos de acordo com as necessidades específicas da empresa, havendo assim um maior emparelhamento e integração com outros produtos Glintt. O grupo Glintt – Global Intelligent Technologies é uma das maiores empresas tecnológicas portuguesas a operar na Europa, África e América Latina, com forte presença em projetos nas áreas da Banca, Telecomunicações, Saúde, Comércio, Indústria e ainda Administração Pública. Neste caso, a Glintt – Healthcare Solutions tem o seu foco completamente direcionado para soluções de *software* na área da saúde.

1.2 Projeto

O Sistema Multiplataforma deve ser uma aplicação que fornece, do lado do servidor, *Web Services*, tais como a possibilidade de alguém enviar um documento assinado para que o Sistema o valide e guarde na base de dados. O sistema também disponibiliza, do lado do cliente, as funcionalidades de assinatura digital, pois estas envolvem a utilização da chave criptográfica privada do médico, e esta não pode estar a ser utilizada do lado do servidor, pois isso obrigaria a que a chave andasse pela *Internet*, o que seria, logo à partida, uma falha básica de segurança.

O projeto consiste sobretudo na parte de assinatura (efetuada pelos médicos) de diversos tipos de documentos, que serão, de seguida, enviados para o servidor onde serão validados e

guardados na base de dados. Posteriormente a este processo, cada vez que algum documento é aberto para ser visualizado, por exemplo, é feita uma verificação de uma possível alteração a esse mesmo documento, para se garantir a sua integridade. Verifica-se, através de métodos de segurança baseados em criptografia de chave pública, se alguém adulterou o documento ou o tornou inválido de alguma forma.

Cada plataforma que o médico utilizar deve ler o seu certificado digital, que está contido no *Smart Card* e é instalado na máquina, uma vez inserido num leitor de cartões apropriado. De seguida é selecionado um documento, para que este seja assinado digitalmente com a chave privada do médico, que é única e pessoal e está mapeada ao respetivo certificado digital, através de uma senha secreta. A assinatura cobre também um *timestamp*, associando uma hora e uma data à operação. Uma vez chegado o documento ao Sistema, este deve então guardá-lo na sua base de dados. A chave privada deve "andar" sempre com o médico e deve ser intransmissível. O *Smart Card* é que contém essa informação secreta, estando esta sempre protegida por um senha. Esta é uma forma de evitar uma possível usurpação de entidade, no acesso à chave privada do médico, pois ao ter-se de inserir uma senha, por exemplo, garante-se que só o médico genuíno é que pode assinar documentos digitalmente, pois, à partida, só ele sabe a senha. Porém, esse cartão tem de ser seguramente pessoal e mais ninguém deve ter acesso a ele, caso contrário outros sujeitos poderiam violá-lo, através da extração da chave privada pelo *hardware*, por exemplo. São técnicas extremamente difíceis, mas não impossíveis.

Concluindo, os requisitos desta dissertação são o médico (utilizador final) poder, a partir de qualquer dispositivo, fixo ou móvel, aceder a uma aplicação *Web* e assinar digitalmente um documento médico, com recurso a um *Smart Card*. O documento após a assinatura deve ser, automaticamente, enviado para o servidor central, verificando-se a sua integridade, e estando tudo validado, este é guardado na base de dados. Logo, deve ser possível assinar digitalmente qualquer documento médico e verificar essa mesma assinatura, posteriormente. Tudo isto em qualquer plataforma e sistema operativo.

1.3 Motivação e Objetivos

Devido à extrema sensibilidade da informação com que se trabalha na área da saúde, é realmente necessário que haja a certeza absoluta de que quem efetua determinadas transações médicas deve mesmo ter permissão e competência para o fazer, isto é, ser realmente a pessoa certa para fazer essas transações. Em áreas que envolvem tratamentos de saúde, marcação de exames ou até de cirurgias e passagem de receitas médicas, deve haver um controlo extremamente eficaz para constatar, efetivamente, que todas as transações devem ser mesmo corretamente efetuadas, não havendo qualquer tipo de engano ou ilegalidade.

Uma das maiores motivações é que o paciente, futuramente, saiba que há um controlo na elaboração dos seus documentos médicos. Todos os documentos que são elaborados sobre a sua

saúde são, garantidamente, elaborados pelos médicos que assinaram digitalmente os documentos.

Com base em todo este controlo necessário surge o maior objetivo de todos, isto é, desenvolver um sistema que controle e valide todas as transações médicas. Visto ser um sistema de *software*, será necessário verificar, em termos de segurança informática, a assinatura digital do médico que fizer a transação.

Esta dissertação visa, principalmente, salientar a grande necessidade de assegurar que todas as transações na área da saúde sejam consideradas íntegras, tendo sido elaborado um sistema distribuído em que todos os envolvidos na área, com os respetivos dados e certificados digitais, tenham a possibilidade de assinar um documento para garantir, *a posteriori*, a sua autoria. Os pacientes ficam também desta forma com a certeza que os seus processos médicos são íntegros, podendo consultá-los e até partilhá-los com outros médicos, tendo a garantia da autenticidade dos mesmos.

O sistema contribui, então, para uma maior organização e um maior controlo numa área muito vasta e com muita sensibilidade e sigilo.

O facto de ser um Sistema Multiplataforma, onde as mais diversas tecnologias e sistemas operativos vão estar envolvidos, e o facto de o sistema central estar preparado para receber inúmeros documentos dos mais variados dispositivos, acrescenta um fator de inovação e uma grande adaptabilidade a qualquer tipo de tecnologia. Acrescenta desta forma um grande nível de abstração que é independente de qualquer tecnologia, sobretudo no lado do cliente, pois terá de responder com sucesso aos mais variados sistemas operativos, adaptando-se de forma a que a plataforma consiga efetuar as assinaturas digitais sem nunca violar qualquer princípio de segurança informática.

1.4 Estrutura do Relatório

Para além da introdução, este relatório contém mais seis capítulos. No capítulo 2, é descrito o estado da arte e são apresentados trabalhos relacionados com a temática da dissertação. No capítulo 3, é referida a tecnologia e as plataformas informáticas a utilizar. No capítulo 4, é apresentada a arquitetura do sistema, sendo esta dividida em vários níveis de abstração que vão desde o mais alto-nível até a informação mais detalhada. No capítulo 5, são descritos todos os pormenores de implementação e todas as contrariedades e dificuldades que foram aparecendo no processo de investigação e desenvolvimento efetuado para chegar ao objetivo proposto inicialmente. No capítulo 6, são descritos os mais variados tipos de testes efetuados para verificar todos os requisitos e validar a sua implementação. No capítulo 7, é descrito e apresentado o trabalho efetuado e também toda uma reflexão sobre o trabalho desenvolvido, incluindo algumas possíveis melhorias e alterações.

Capítulo 2

2. Revisão Bibliográfica

Neste capítulo é descrito o estado da arte e são apresentados trabalhos relacionados de forma a mostrar o que existe no mesmo domínio e quais os problemas encontrados. É referido todo o trabalho e investigação publicada relacionada com este assunto, para que se conheça com detalhe a forma como esta temática foi abordada e analisada em casos idênticos.

Para além de trabalhos relacionados foram estudados e analisados as principais ferramentas tecnológicas utilizadas neste tipo de casos, e plataformas semelhantes, de forma a guiar esta dissertação em futuras escolhas.

2.1 Introdução

Neste capítulo iremos ver o que se utiliza mais em termos de infraestruturas em segurança informática que evitem fraudes de integridade. O estudo parte de artigos onde são abordadas técnicas o mais generalizadas possíveis até ao estudo de artigos mais concretos e passíveis de abordagem nesta implementação do sistema multiplataforma.

Em segurança, há normalmente dois tipos de abordagem criptográfica: ou se utiliza um sistema de chaves simétrico, onde há um segredo e uma chave partilhada entre os diversos intervenientes do sistema; ou então é utilizado um sistema de chaves assimétrico, onde cada entidade tem um par de chaves (chave pública e chave privada), sendo a chave privada apenas do conhecimento da respetiva entidade, não podendo, assim, ser partilhada com mais ninguém. A segunda abordagem, à partida, interessa mais ao contexto em causa, pois não se quer que haja segredos partilhados, mas sim que haja sigilo e que cada médico tenha algo que é, única e exclusivamente, só seu.

2.2 Desenvolvimento de uma infraestrutura de chaves públicas

Nos últimos anos vem-se acentuando a necessidade de validar corretamente e com confiança as mais variadas transações na área da saúde. Esta é uma área que envolve o maior sigilo e a maior confidencialidade quanto à informação, pois é da maior vulnerabilidade e deve ser devidamente preservada. Vamos então explorar um pouco mais este tipo assimétrico de criptografia.

Cada vez mais, organizações variadíssimas, tais como bancos, operadoras de telecomunicações, companhias petrolíferas, correios, farmácias e agências governamentais já utilizam sistemas de chave pública (Khan 1998, 18).

Em sistemas do tipo *Public-Key Infrastructure* (PKI) são necessárias assinaturas digitais e também gestão de chaves seguras para que seja possível suportar encriptação e autenticação. De facto, duas entidades que estão a milhares de quilómetros de distância e nunca se encontraram antes necessitam de uma terceira, uma *Certification Authority* (CA), em que ambos confiem. Neste caso, as assinaturas digitais, que é o núcleo da dissertação, são importantes para a estruturação de PKIs.

A primeira análise que fazemos ao desenvolver uma infraestrutura de chave pública é ver se realmente é necessária, devido ao facto de a sua utilização ser computacionalmente bastante pesada em comparação, por exemplo, com os melhores algoritmos de chave simétrica. Temos que considerar então primeiro, se não faz mais sentido um sistema de chaves simétricas, devido ao facto de ser mais eficiente e mais leve em termos de computação ou então até de considerarmos uma solução híbrida. Devemos basear a decisão, tendo em conta a lógica de negócio e os requisitos de segurança que daí advêm. Temos que fazer uma análise de risco na qual se identificam as ameaças e as vulnerabilidades possíveis do sistema e como estas podem ser evitadas (Khan 1998, 19).

Apesar de mais pesados computacionalmente, os sistemas assimétricos, isto é, os que utilizam sistemas de chaves públicas, são mais aplicáveis neste caso, porque dispensam chaves e segredos partilhados. A chave privada deve então estar sempre junto à entidade, sendo apenas esta a poder aceder àquela. A não ser assim, e no caso da saúde seria o mesmo que o médico confiar a sua chave pessoal a uma entidade central, permitindo que esta se pudesse fazer passar por ele a qualquer momento. Mesmo tendo em conta que é uma entidade em que ele confia, por princípio de segurança, não deve ter acesso à chave privada do médico. Acentuamos, então, a importância de utilizar uma infraestrutura deste tipo (Khan 1998, 20).

Optando então por uma assinatura digital com recurso a criptografia assimétrica, é necessária uma PKI, que implica a necessidade de haver uma CA. Esta CA serve para mapear uma entidade ou um utilizador à sua chave pública criptográfica. Isto é conseguido através de uma assinatura digital por parte da CA. Esta assinatura é feita com criptografia de chaves públicas e contém o nome do utilizador, chave pública e outros dados relevantes. Esta CA não é o foco de desenvolvimento, nesta dissertação, pois já estamos a assumir que já existe, dentro do

contexto hospitalar, e todos os mapeamentos entre a entidade e a chave pública já estão guardados nesta autoridade.

Então, a assinatura digital depende da existência de uma CA, mas esta também depende de assinaturas digitais para funcionar com consistência. Esclarecendo um pouco mais, uma assinatura digital é uma forma de autenticar informação digital. É tratada como uma analogia à assinatura física em papel, pois existem semelhanças, mas também diferenças importantes. É confundido com assinatura eletrónica, por vezes, mas essa é um método não criptográfico para validar assinaturas físicas manuscritas. Já a assinatura digital utiliza criptografia e providencia uma prova inegável de que uma certa mensagem veio do emissor. São efetuadas encriptações às mensagens que são facilmente verificadas com desencriptações posteriores. A assinatura digital garante autenticidade (o recetor pode confirmar que a assinatura foi feita pelo emissor) e integridade (qualquer alteração da mensagem faz com que a assinatura não corresponda mais ao documento), sendo que estes dois conceitos de segurança informática são os que mais pretendemos assegurar com este sistema multiplataforma de integridade nas transações médicas. Mas para haver chaves públicas e privadas que se possam relacionar matematicamente, cifrando e decifrando textos corretamente é preciso haver uma CA que se responsabilize por atribuir as chaves a cada uma das entidades. Porém, esta CA tem também uma chave pública e outra privada. Para que as entidades envolvidas no ambiente criptográfico sejam certificadas e reconhecidas umas pelas outras, o CA atribui um certificado digital e uma chave privada a cada uma delas, sendo que este certificado está assinado digitalmente com a chave privada da CA como podemos ver na figura 1.

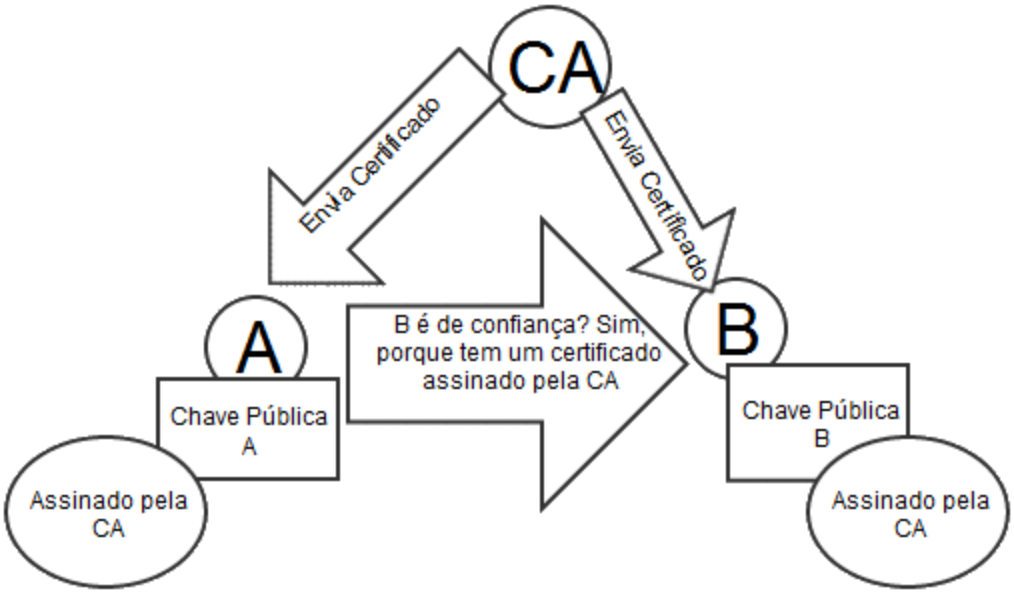

Figura 1: CA a certificar entidades

Assim todas as outras entidades podem facilmente testar a veracidade das entidades envolvidas, utilizando a chave pública da CA para esse efeito. Ou seja, desta forma CAs e assinaturas digitais estão interligadas, tendo vindo a ser utilizadas desta forma nos últimos anos nos mais diversos sistemas informáticos criptográficos.

2.2.1 Desenho da Arquitetura

É, normalmente, insustentável ter uma CA que gere e trata de autenticar utilizadores (médicos no caso), também assinar certificados, no caso de grandes números de utilizadores utilizarem a infraestrutura, o que acontece também no caso do ambiente desta dissertação. Tendo em conta os possíveis milhares de acesso que os médicos irão fazer para enviar documentos assinados digitalmente, impomos, em situações deste género, uma infraestrutura mais distribuída.

Utilizamos, usualmente, uma «... *Registration Authority* (RA) que é a responsável por autenticar os utilizadores finais, comprovando as respetivas credenciais. Logicamente residem entre os utilizadores finais e a CA, funcionando como camada intermédia, sendo responsáveis por identificar a identidade dos utilizadores que pedem certificados. Naturalmente, há que verificar com cuidado se a informação que o utilizador está a fornecer é realmente correta e se é válida. A forma como o utilizador é autenticado é um protocolo definido entre a RA e a CA e depende da utilização do certificado pedido...» (tradução pessoal de (Khan 1998, 21));

Existe na mesma a CA, que é a entidade que se irá preocupar com o facto de que os certificados foram criados dentro da política acordada previamente e que o pedido já foi submetido por uma RA. Uma vez feitas estas verificações, é necessário gerar o certificado e passá-lo a uma terceira componente, que é o diretório de serviços, para publicação e também para devolver o certificado à RA apropriada (Khan 1998, 21).

Por fim o diretório de serviços, que também costuma existir nesta arquitetura distribuída, é responsável por guardar os certificados de chaves públicas e as *Certification Revocation Lists* (CRLs). Este serviço não tem de ser de confiança obrigatoriamente, pois tanto os certificados como as CRLs são assinadas digitalmente pela CA, logo qualquer modificação não autorizada vai ser detetada seguramente. Por exemplo, os cartões de identificação eletrónica, nos mais diversos países, têm um serviço de CRL, pois, periodicamente, é necessário a renovação destes cartões por motivos de segurança. Os certificados digitais contidos nestes cartões são revogados e são criados uns novos, uma vez obtidos os novos cartões. Também na área da saúde os médicos em causa irão renovar os seus cartões e os seus certificados digitais por motivos de segurança informática.

Quanto à PKI base em si, costuma ter, normalmente uma CA raiz, que representa o topo de uma cadeia de confiança, isto é, a chave pública da CA é instalada em sistemas dos utilizadores finais e, consequentemente, utilizada para verificar outros certificados. Neste caso a proteção da

chave privada da CA raiz é nuclear e crítica e deve ser utilizada o menos possível. Quem só utilize uma única CA (e não uma cadeia de CAs) deve ter pelo menos mais uma camada na hierarquia de CAs, precisamente para que não estejamos sempre a utilizar a chave privada da CA raiz, evitando assim expor a parte mais sensível do sistema (Khan 1998, 23).

Neste tipo de infraestrutura, cuidados como a atualização da chave da CA raiz são determinantes para que utilizemos convenientemente e com a maior segurança e confiança possível. Nesta atualização deve haver sempre uma série de passos a seguir, ou seja, há que primeiro gerar o par de chaves novo (a chave pública e a chave privada), de seguida, criar um certificado que contenha a chave pública da antiga CA raiz assinada com a nova chave privada. Posteriormente, é criado um certificado contendo a chave pública da nova CA raiz assinada com a antiga chave privada, ou seja, o contrário do passo anterior. Para além disto, normalmente, é criado ainda um terceiro certificado contendo a chave pública da nova CA raiz assinada com a nova chave privada. Para finalizar este processo, é necessário publicar e exportar a chave pública da nova CA raiz para que as entidades finais possam adquiri-la. Desta forma a chave privada da antiga CA raiz não é mais necessária. Todo este processo é necessário, porque estamos a mexer com dados muito sensíveis, cuja integridade deve ser preservada a todo o custo (Khan 1998, 28 - 29).

De realçar novamente que as referências à CA ao modo de como esta deve estar construída, servem para explicar como se desenrola todo o processo de certificação das autoridades. Esta não será desenvolvida, porém é relevante, pois em toda a implementação efetuada, assumimos que a CA está bem desenvolvida, bem equilibrada e distribuída, em termos de eficiência, e que garante que todos os médicos estão devidamente registados, podendo ser considerados válidos aquando a verificação das suas assinaturas digitais e que a gestão da revogação dos certificados é também devidamente efetuada.

2.2.2 Protocolos necessários

Alguns protocolos utilizados neste tipo de infraestrutura devem ser tidos em conta, tais como suportar e fornecer um serviço de recuperação de chaves.

Poderá fazer sentido que entidades devidamente autorizadas possam recuperar os textos previamente encriptados, caso a chave de desencriptação não esteja disponível por algum motivo. Ou no caso das assinaturas digitais, poderá ser necessário recuperar uma chave pública para se validar a assinatura efetuada com a chave privada. A maior parte das vezes a chave pública é também acrescentada ao documento na assinatura, por via de um certificado digital, para que seja utilizada para validar o documento. Tem de haver uma entidade que contenha todas as chaves públicas de todos os médicos e, para a eventualidade de não se conseguir decifrar o que o médico cifrou, utilizamos, normalmente, um *Key Recovery System* (KRS). Um KRS permite a pessoas autorizadas recuperar os originais de dados encriptados quando a chave de desencriptação não está, por qualquer motivo, disponível. É um elemento auxiliar à CA

muita das vezes, servindo como uma forma de redundância de informação. Existem variadas técnicas que recuperam chaves, mas todas têm em comum o facto de recuperar a chamada "chave-alvo". Esta chave pode servir para desencriptar a informação, direta ou indiretamente (Gupta and Matyas Jr 2000, 56).

Tipicamente, existem dois tipos de técnicas de recuperação de chaves: encapsulamento da chave e *backup* da chave. A técnica de *backup* utiliza os recursos de pelo menos um *Key Recovery Agents* (KRAs), que detêm chaves, ou parte/pedaços delas, que dizem respeito aos seus assinantes. Já na técnica de encapsulamento os KRAs não têm as chaves que pertencem aos assinantes. É sim criada uma estrutura que contém informação que vai permitir ao(s) KRA(s) ajudar na recuperação da "chave-alvo".

Em suma, no contexto desta dissertação, pode ser relevante que exista um KRS, pois podem existir alturas em que é necessário desencriptar a informação que os médicos encriptaram (validando assim as assinaturas digitais por eles efetuadas), no caso da parte que gere todas as chaves públicas falhar, momentaneamente, ou até falhar de vez. Facilmente vemos que qualquer KRS se integra bem com a PKI, o que torna ainda mais interessante a sua possível abordagem neste contexto. Porém, realçamos novamente que este não é o foco principal do problema, logo é assumido também que a entidade que desenvolve a PKI e, possivelmente, o KRS, gere todos estes processos devidamente, o que dá um suporte fiável e consistente para que as assinaturas digitais funcionem corretamente.

Voltando então ao ponto fulcral da dissertação, chegamos ao estudo da assinatura digital em si. Esta assinatura simula a assinatura humana, mas em vez de ser uma digitalização da assinatura manual é um conjunto de *bits* que formam uma cadeia de informação única. Cada utilizador tem uma chave secreta para assinar, e, de seguida, publica para todos a sua chave pública correspondente. Desta forma, o utilizador pode assinar com a sua chave secreta, e qualquer outro utilizador, com a chave pública do primeiro, pode verificar se realmente aquela mensagem foi assinada por esse mesmo utilizador.

Neste caso há diversos ataques, que têm de ser tidos em conta se realmente quisermos que os médicos transacionem os mais variados documentos seguramente. Um deles é o *Key-Only Attack* onde o atacante sabe apenas a chave pública e outro é o ataque de mensagem, onde o atacante consegue examinar algumas assinaturas que correspondem a mensagens que ele saiba serem daquele utilizador. Dizemos que uma assinatura está inutilizada, quando um adversário consegue computar a chave privada do assinante; quando o atacante encontra um algoritmo de assinatura que funcione igualmente ao que foi utilizado para assinar; quando o atacante consegue forjar uma assinatura de uma mensagem em particular; e ainda quando o atacante pode conseguir forjar uma assinatura para pelo menos uma mensagem. Para evitar este tipo de ataques e tornar as chaves realmente seguras, é recomendado utilizarmos o esquema de assinaturas Guillou-Quisquater que se baseia nos mesmos fundamentos que o algoritmo de RSA na questão dos números primos (Zhang 2003, 27 e 31). Este esquema torna mais difícil o ataque e a invalidação do processo de assinaturas digitais.

2.2.3 Desenho de uma infraestrutura de chaves públicas móvel

O conteúdo desta secção baseia-se principalmente em (Ray and Biswas 2011, 297 e 298).

As estatísticas demonstram que os dispositivos móveis são cada vez mais utilizados e, consequentemente, a variedade de serviços eletrónicos que dão suporte a esses dispositivos está também a aumentar. A segurança é uma das maiores preocupações para os utilizadores de dispositivos móveis que utilizam serviços eletrónicos através da *Internet* sem fios. Tal segurança deve basear-se em garantir quatro pontos fundamentais neste tipo de transações: confidencialidade; integridade dos dados; autenticação e a não rejeição. Em ambientes com fios vimos já, anteriormente, que a PKI gera certificados de chave pública, mapeando a chave pública com o seu dono e transmite-a de uma forma segura e fiável utilizando um método de criptografia com chaves públicas. Do mesmo modo, em cenários sem fios, um certificado de chaves públicas pode ser atribuído a um utilizador do dispositivo móvel. O utilizador do dispositivo pode usar o certificado para se autenticar e criar, assim, um canal seguro para transacionar seguramente.

Foi proposto em 2008 um protocolo sem fios adequado a dispositivos móveis, onde um dispositivo pode pedir um certificado de chaves públicas a uma CA, que, posteriormente, faz a autenticação do utilizador e a verificação de *Proof of Possession* (POP) da chave privada do dispositivo, que corresponde à sua chave pública. Após tudo isto, a CA publica o certificado no seu diretório ou na WWW e fornece um certificado ou um *Uniform Resource Locator* (URL) ao utilizador.

Para eliminar dificuldades, como as baixas taxas de processamento e pouca memória de alguns dos dispositivos móveis atuais, consideramos normalmente a utilização de um *Mobile Home Agent* (MHA) que se baseia em PKI móvel. O MHA guarda toda a informação do dispositivo e realiza todas as operações para obter o certificado do dispositivo através da CA. Consideramos também utilizar uma RA para minimizar a carga da CA relativamente às operações de autenticação do utilizador e verificação da POP não sobrecarreguem a CA. Desta forma, a CA verifica apenas se o pedido passou ou não pela RA e, em caso afirmativo, então cria o certificado de chaves públicas do dispositivo móvel, assina-o e publica-o no seu diretório, fornecendo o URL do certificado ao MHA, e este terá só de enviar o URL do certificado para o dispositivo. O utilizador do dispositivo utiliza aquele URL para se autenticar e certificar-se que utiliza um canal seguro com ligação a um servidor para as mais variadas transações. Neste caso, a RA estaria separada da CA e teria como função autenticar as entidades, distribuir *tokens* (pedaços de informação gerados aleatoriamente que identificam uma entidade univocamente), gerar relatórios de anulações, mapear nomes, gerar chaves e arquivar pares de chaves. Cada RA tem de ser certificado pela CA e tem de ter chaves privadas próprias para assinar digitalmente.

O certificado mais utilizado, e designado pela *Internacional Telecommunication Union* como o mais adequado para descrever a informação de uma maneira estruturada, é o X.509. A versão três do X.509 tem diversos campos como o nome do sujeito e do emissor, uma chave

pública associada ao sujeito, um período de validade e ainda um campo com o número identificador do algoritmo de assinatura do certificado. Esta versão de certificados digitais foi a escolhida para utilizarmos nesta dissertação, pois a maior parte dos certificados digitais hoje em dia são deste formato. Os cartões de teste irão também conter certificados digitais neste formato.

Para esta secção de conclusão foi referenciado (Ray and Biswas 2011, 297 e 298).

Este tipo de infraestruturas móveis são extremamente importantes na utilização de aplicações *Web* que utilizem protocolos do tipo *Secure Sockets Layer* (SSL) / *Transport Layer Security*. Uma das abordagens tentadas na definição da arquitetura e da implementação da dissertação utiliza SSL/TLS, como iremos poder verificar mais à frente nos capítulos Arquitetura e Implementação. Daí termos como base o funcionamento corrente de como funciona uma PKI em ambientes com sistemas operativos móveis. Em princípio a PKI que estará por detrás do ambiente hospitalar não será móvel, porém toda a interação e protocolos de segurança na utilização das plataformas móveis seguem os mesmos princípios aqui revistos.

2.2.4 Utilização de PKI em comércio eletrónico

Para esta secção utilizou-se como base o texto de (Hernandez-Ardieta, Gonzalez-Tablas, and Alvarez 2008, 310-311).

O grande crescimento do comércio eletrónico nos últimos anos vem permitindo que um grande número de empresas possa comprar e vender praticamente qualquer produto na *Internet*. O grande problema é que, normalmente, o vendedor tem informação consideravelmente sensível do comprador, tais como, a sua informação do cartão de crédito e afins, e o contrário não acontece. Tendo em conta este cenário, começamos a utilizar PKIs para gerar um protocolo de troca de mensagens justo e baseado em políticas de segurança, que garanta também ao comprador a possibilidade de poder ou não confiar nas regras e nas políticas definidas pelo vendedor. Ou seja, também em transações monetárias se recorre a infraestruturas e a técnicas criptográficas, o que dá mais um reforço à sua utilização nas transações médicas.

Um dos protocolos utilizados neste caso pressupõe que todo o desenho seja baseado em políticas de assinatura e que seja criado um conjunto de regras para gerar e validar assinaturas eletrónicas, tais que quaisquer assinaturas possam ser determinadas se são válidas ou não, numa transação em particular. Todo este protocolo é seguido por todas as entidades envolvidas na transação eletrónica e serve, sobretudo, para assegurar a sua justiça, segurança e cumprimento de prazos.

Para servir de intermediário entre o vendedor e o comprador utiliza-se também uma *Trusted Third Party* (TTP), que é um sistema independente, da confiança de ambos os intervenientes na transação, e que atua com uma conduta justa e apropriada durante toda a execução do protocolo. Toda a troca de informação, no decorrer de uma transação eletrónica, irá recorrer a elementos fundamentais que a PKI fornece, desde a base do chaveamento público-privado até a uma espécie de CA que será o intermediário, isto é, a TTP. A troca segura de

informação vem sendo cada vez mais utilizada no comércio eletrónico, e é dos maiores casos de utilização que as PKIs têm na WWW. Esta técnica baseia-se em assinaturas digitais e eletrónicas, pelo que as eletrónicas seriam também uma possível abordagem nas transações médicas. Porém, as que conferem realmente segurança inequívoca, que qualquer alteração a um documento faça com que a assinatura não corresponda mais ao documento e onde o recetor pode confirmar que a assinatura foi efetuada pelo emissor, são as assinaturas digitais.

2.3 Partilha segura de dados médicos

Esta primeira secção da partilha segura de dados médicos foi baseada tendo em conta (Ruoyu, Gail-Joon, and Hongxin 2012).

Nos sistemas de saúde modernos, cada vez mais os dados médicos eletrónicos estão a ser migrados para a *Cloud*. A *Cloud* refere-se à ideia de utilizarmos, em qualquer lugar e independentemente da plataforma, os mais diversos programas por meio da *Internet* sem que estes estejam instalados nos nossos computadores. Há uma necessidade de reduzir custos e em vez de construir e manter infraestruturas de dados específicas e adequadas às necessidades de cada um, recorre-se à *Cloud* e, desta forma, há um menor custo e uma melhor operacionalidade. Contudo, a adoção da *Cloud Computing* em sistemas de saúde levanta cuidados e desafios de segurança, tais como, a autenticação, gestão de identidades, controlo de acessos e afins.

Tal como no contexto desta dissertação, os pacientes têm diversos cuidados médicos e ligação a várias especialidades médicas e, consequentemente, todos os dados necessitam de estar devidamente guardados para que possam ser revistos e geridos por eles, podendo partilhar os seus dados com outros sujeitos ligados à saúde, se assim o quiserem.

Uma política proposta é, por exemplo, os pacientes darem acesso dos seus dados eletrónicos médicos a indivíduos devidamente identificados. O acesso pode ser dado a médicos, individualmente, selecionados pelo seu nome, por exemplo, ou então a médicos por áreas, tais como médicos que sejam dentistas, oftalmologistas, etc. «Para proteger melhor a privacidade de um paciente, quando este partilha a sua informação médica, é especificada na política de autorização as razões ou os motivos que o levaram a querer dar acesso a determinado sujeito ou conjunto de sujeitos.» (tradução pessoal). O módulo de segurança proposto neste artigo define então três módulos: o de autenticação, para assegurar que só os utilizadores verdadeiros têm acesso ao sistema; o de controlo de acessos, para controlar o acesso dos utilizadores aos registos médicos eletrónicos; e ainda uma auditoria que mantém *logs* de todo o sistema, permitindo assim a sua posterior análise e verificação de possíveis não conformidades. Já o módulo de gestão da política de segurança é dividido em mais dois módulos: o de especificação da política que fornece a capacidade para os pacientes especificarem as suas políticas de controlo de acesso; e também o módulo de cumprimento da política, que assegura que as políticas de

segurança, quando recebem pedidos de acesso aos dados eletrónicos, por parte dos utilizadores, geram resultados de autorização para o módulo de controlo de acessos.

Concluindo, é um artigo que acrescenta boas ideias de como os dados devem estar guardados e como devem ser acedidos, pois define uma política acertada com cuidados de sigilo que a área da saúde deve conter. Estando os dados centralizados, possivelmente acedidos por qualquer sujeito, é necessária esta definição de política de acesso, que pode fazer sentido utilizar na dissertação. Não sendo o foco principal, esta é uma solução a adotar quando os documentos assinados pelos médicos forem realmente integrados com as soluções e os serviços centrais hospitalares. Os documentos, mesmo que assinados devem estar protegidos de acessos indevidos e não-autorizados.

A partir daqui, o texto é escrito tendo em conta (Fernandez-Aleman et al. 2013).

Sentimos, cada vez mais, que os dados médicos e hospitalares devem estar mais organizados e, por isso, há a necessidade de digitalizar todos estes registos. Tendo em conta a importância que estes têm na nossa sociedade, o facto de estarem digitalizados ajuda no controlo e na organização dos mesmos, reduz os custos (visto que espaço digital é mais barato que espaço físico) e, sobretudo, proporciona uma mobilidade tremenda, pois os dados podem ser acedidos através de qualquer dispositivo com ligação a rede externa. Para tornar tudo isto possível é necessário satisfazer requisitos em termos de armazenamento dos dados, resistência a falhas, grande disponibilidade e ainda manter sempre uma grande consistência em termos de políticas de segurança.

As três componentes de segurança informática mais importantes são a confidencialidade, a integridade e a disponibilidade. O sigilo na área da saúde é fundamental e este «processo deve assegurar que a informação é acessível apenas àqueles que estão autorizados a acedê-la» (tradução pessoal). A integridade é outra componente fundamental nesta área, pois é esta que assegura que a informação é precisa e não foi modificada por uma entidade não autorizada. Também a disponibilidade é um fator a ter em conta, pois os registos médicos devem estar disponíveis para consulta ou alteração a qualquer momento.

Estudos recentes demonstram que em cada ano, cerca de vinte e cinco milhões de autorizações indevidas nos Estados Unidos da América aconteceram, o que levou à revelação de registos de saúde pessoais e confidenciais. Uma das técnicas utilizadas até agora para proteção é a técnica do uso de um pseudónimo, que permite a terceiras entidades aceder a dados da saúde dos pacientes, sem que se descubra os dados pessoais do paciente. Cada paciente tem um identificador que é um *token* único. A partir do identificador é normalmente muito complicado reverter e encontrar o paciente a que aquele *token* diz respeito. Muitos destes métodos utilizam o *Advanced Encryption Standard* (AES) para gerar o pseudónimo. A integridade nestes casos é também importante nesta técnica do pseudónimo, pois há que ter a certeza que o pseudónimo não é alterado. É necessário para além disto encriptar dados, como estes pseudónimos, chaves e outros atributos para aumentar a segurança. Neste caso, as técnicas variam entre as de chave pública e as de segredo partilhado, sendo que, creio, apesar de mais pesadas

computacionalmente, as de chave pública são mais apropriadas, já que estamos a falar de ambientes muito abertos com muita diversidade de acessos.

Em relação às comunicações, usualmente, utiliza-se uma CA e é tudo encriptado através de SSL, via (TLS) ou através de outros protocolos de segurança.

No controlo de acesso, é proposto neste artigo (Fernandez-Aleman et al. 2013, 545) novamente um esquema de assinaturas digitais baseados numa PKI. É utilizado um certificado que gere os direitos da conta do utilizador e mapeia um par de chaves utilizado para a sua autenticação, e este é mapeado ao identificador do utilizador e endereço de e-mail, por exemplo. Outros mecanismos de acesso são o normal: *login/password, login/password* com certificados digitais, *Smart Cards* e outros.

Por último, é necessário especificar corretamente quem define os papéis que cada sujeito vai ter no sistema. Utilizamos papéis baseados em utilizadores, individualmente, e em áreas, para que seja mais fácil examinar, posteriormente, quem teve permissão para aceder a determinada informação. Para além disso, cada papel deve ter etiquetado um tempo de início e de fim para que se saiba a altura de utilização no sistema. Em termos de prevenção e deteção, estas técnicas são bastante utilizadas, pois os sistemas podem conseguir-se reajustar e alterar o seu comportamento para que tais falhas não voltem a ocorrer.

Em suma, na área da saúde a maior preocupação que existe é com o controlo de acessos, em termos de segurança informática. O que existe maioritariamente são cuidados deste tipo. O objetivo desta dissertação é dar um complemento a esta área com a integridade dos processos e das transações efetuadas pelos médicos. Até aqui o maior foco foi o sigilo (que é naturalmente dos mais, senão mesmo o mais importante nesta área), porém não conseguimos garantir com o controlo de acessos que certo e determinado documento prescrito pelo médico foi, seguramente, elaborado por ele. Daí acreditarmos que o complemento de ambos os aspetos de proteção possam dar um novo sentido de segurança e confiança à área da saúde, estando tudo confidencial e com os devidos acessos e permissões estabelecidas na mesma, mas com garantia de que qualquer prescrição médica sem controlo de origem, sendo as fraudes devidamente detetadas.

2.4 Utilização de Smart Cards

Estudemos agora um instrumento auxiliar a todo este processo: o *Smart Card*. Este é um cartão que tem, normalmente, dois objetivos principais: a identificação pessoal e o pagamento de transações, como, por exemplo, os cartões bancários. Contém espaço de armazenamento e é capaz de um processamento superior a outros cartões, visto que tem memória e um microprocessador. Este é um cartão que utiliza mecanismos de segurança eficazes de proteção, dos dados eletrónicos que contém dentro dele, e é, por isso, um elemento que pode ser considerado muito importante na autenticação dos médicos, para que não haja um roubo de

identidades. Acrescentando a todas estas funcionalidades, têm sido cada vez mais desenvolvidos *Smart Cards* que contêm também um certificado digital e uma chave privada, associada a esse certificado, próprios para operar com assinaturas digitais. Foi, por isso, efetuado um estudo bibliográfico que nos apresenta as diversas áreas de utilização do *Smart Card* e sobretudo como os dados internos são lidos. Tendo como base este estudo, a abordagem no contexto da dissertação vai basear-se nestas metodologias já imensamente utilizadas e testadas.

2.4.1 Na Autenticação e Acesso através da Internet

Esta secção inicial de autenticação é baseada em (Verschuren 1998).

Na WWW, hoje em dia, estamos a utilizar métodos de autenticação como filtros através de endereços *Internet Protocol* (IP), ou métodos mais comuns como o *username/password*. Há, no entanto, outras abordagens como a utilização de *Smart Cards* que combinam a parte física e visual do cartão com os dados eletrónicos contidos nele. Estes *Smart Cards* são utilizados para efetuar autenticação forte em serviços *on-line*. Existem protocolos que, com um *Smart Card*, um leitor apropriado de *Smart Cards*, um computador e um navegador *Web,* permitem a um utilizador autenticar-se num servidor e aceder a dados próprios privados.

«O cartão em si tem uma arquitetura semelhante a uma disquete MS-DOS. O cartão pode ter até cerca de vinte níveis de segurança, cada um com chaves correspondentes. Isto permite o uso de diferentes aplicações e independentes entre si… As chaves são utilizadas para assegurar privacidade, integridade dos dados, encriptação de comunicações e para permitir autenticação. Certos dados e campos no cartão devem ser lidas apenas por aplicações externas, após o utilizador fornecer o seu código (*Personal Identification Number* (PIN), por exemplo).» (tradução pessoal).

Existem já formas de ler os dados do *Smart Card* através de um navegador *Web*. O proposto neste artigo é o modelo ISI:

«1 – Primeiro, o cliente utiliza o navegador para selecionar e carregar a página inicial apropriada, que contém informação generalizada. Nesta página, o cliente primeiro seleciona o tipo de cartão e depois um botão para pedir acesso a dados protegidos. Este botão seleciona o primeiro *script* de *Common Gateway Interface* (CGI) que está alojado no servidor. 2 – De seguida, o *script* começa um procedimento para iniciar uma conexão SSL. Quando a ligação SSL estiver pronta, um *applet* Java será selecionado e carregado do servidor para o lado do cliente (navegador). Este *applet* será guardado na parte dinâmica do navegador. 3 – O *applet* começa a correr automaticamente. É gerado um número aleatório no servidor que irá servir para autenticação. 4 – Quando o *applet* recebe o número envia alguns comandos para a *Application Programming Interface* de Java *ChipCard*. O *applet* carrega a informação que foi enviada para o *Smart Card*... O número aleatório é enviado para o cartão e os dados, juntamente com um *Message Authentication Code* são enviados para o *applet* pelo *Smart Card*. 5 – Quando todos os comandos são executados, é construído um URL e o navegador carrega-o. Antes do URL ser

enviado, o servidor verifica se todos os parâmetros estão corretos. Para verificar o *Authentication Code*, um processo criptográfico tem de ser iniciado. 6 – Se a chave do servidor for a mesma chave do cartão, e se os dados ou código de autenticação não forem alterados durante a transferência, os códigos calculados e o recebido têm de ser iguais. Isto significa que a autenticação foi efetuada com sucesso e então o dono do cartão tem acesso aos dados pedidos.» (tradução pessoal).

Este é um exemplo que pode ser explorado e abordado no contexto da dissertação. Em princípio, para o caso de o médico perder o seu *Smart Card,* se este tiver uma espécie de PIN de acesso ou outro código garante-se que um terceiro não consegue utilizar o *Smart Card*.

A partir desta segunda secção, o resto do texto passa a ser baseado em (Chan 2000) e apresenta mais uma forma de utilização do conteúdo dos *Smart Cards* na WWW. Estamos a explorar a sua utilização na WWW, pois a ideia é precisamente que este tipo de cartões seja utilizada na multiplataforma que é uma aplicação *Web*.

Tendo em conta que hoje em dia toda a informação eletrónica de um paciente está dividida entre sistemas de informação de hospitais/clínicas diferentes, os registos do paciente deixam de estar localizados num único sítio, evitando poder aceder a toda a informação do paciente de uma vez só. Com um *Smart Card* qualquer pessoa pode trazer consigo a sua informação pessoal. Com a informação contida no cartão, em qualquer local, poderiam facilmente aceder à informação do paciente para que fosse possível verificar o seu histórico médico. Usualmente, é utilizado uma *interface front-end* que consegue ler a informação do cartão através de um navegador *Web*. Utiliza-se então uma *Smart Card – Web Gateway Interface* que providencia uma comunicação baseada em *Hypertext Transfer Protocol* (HTTP) entre o cliente e o conteúdo do *Smart Card*. A informação contida no cartão, normalmente está contida em *Extensive Markup Language* (XML), mas pode também conter ficheiros *HyperText Markup Language* (HTML), imagens e outros documentos. Para acedermos ao XML com a informação de uma entidade, o *Smart Card* apresenta segurança de acessos tanto internamente como externamente. Externamente, guardamos a informação que identifica a entidade, e, internamente, estão os dados mais sensíveis que contêm encriptação, que permite haver múltiplos níveis de acesso.

O conteúdo do *Smart Card* é normalmente desenvolvido na tecnologia Java *Card* que suporta desenvolvimento de aplicações baseadas na linguagem de programação Java de uma forma segura, estando apta a correr precisamente em dispositivos com capacidade limitada de memória e capacidade de processamento. Mas «apesar da flexibilidade e robusteza do Java *Card* em escrever em *applets*, definindo assim comunicações entre o cartão e o cliente, é ainda assim necessário o uso de um *Application Protocol Data Unit* (APDU), como protocolo básico para troca de mensagens e respostas.» (tradução pessoal). Basicamente este protocolo representa um servidor que lê e interpreta pedidos HTTP dos navegadores dos dispositivos, que querem aceder ao conteúdo do cartão, e gera respostas apropriadas. Normalmente, um leitor de cartões está ligado a um computador com este protocolo a fornecer um acesso baseado em *Web* entre o navegador e o *Smart Card*.

2.4.2 No comércio eletrónico

Os *Smart Cards*, são também cada vez mais utilizados no comércio eletrónico que envolve transações com dinheiro, o que garante (à partida) que é um meio com segurança, que apresenta confiança na sua utilização.

Os *Smart Cards* têm vantagens neste tipo de transações, pois contêm chaves criptográficas, certificados digitais, perfis e outros dados que permitem que o conteúdo do cartão esteja protegido contra roubos, uma vez que os ficheiros são encriptados.

Também a capacidade de armazenar e proteger os dados de transações com dinheiro num pequeno "compartimento" dentro do cartão é uma das razões que leva ao seu uso neste ambiente. Este facto reduz comunicações de rede, uma vez que não precisa de ser autenticado nem identificado pela base de dados de um determinado banco. Ainda o facto de poder conter impressões digitais digitalizadas, é considerada uma vantagem enorme neste tipo de transações (Mcelroy and Turban 1998).

Concluindo, é um cartão utilizado em diversas áreas que envolvem muita segurança e muita proteção de dados, tal como o comércio eletrónico, logo apresenta-se como um dos métodos mais interessantes para que o médico armazene a sua informação de uma forma segura e portável.

2.5 Plataformas existentes

Em termos de plataformas semelhantes já existe na Estónia um cartão de identificação que contém um *chip* com ficheiros embebidos que utilizam encriptação de chaves públicas de 2048 bits. Este cartão é utilizado regularmente como: um cartão de identificação nacional; um cartão de seguro de saúde nacional; uma prova de identificação quando um utilizador tenta autenticar-se num banco através da *Web*; também é utilizado como um cartão de transportes públicos pré-pagos; para assinaturas digitais; para votar; para aceder a registos médicos que estejam contidos em bases de dados governamentais; e para levantar prescrições eletrónicas. Este é o cartão de identificação nacional mais desenvolvido comparativamente a qualquer outro país do Mundo. Com a assinatura digital é possível detetarmos alteração de dados, pois qualquer alteração invalida tanto a assinatura como o *timestamp* da mesma. Neste caso, é também utilizado uma PKI a nível nacional.

É um cartão muito parecido com o cartão de cidadão português, que contém informação pessoal para que a pessoa seja facilmente identificada, e contém também dois certificados e as suas chaves privadas associadas, que estão protegidas por códigos PIN. Um dos certificados é utilizado para autenticação e o outro para assinaturas digitais. Cada um deles tem a sua chave privada associada, como já foi referido, e há um código PIN para cada aceder a essa mesma chave de cada um dos certificados. Quando um cidadão pretende suspender ou anular um dos

certificados este nunca mais é utilizado para nenhum outro. Para além disso, nenhuns dados públicos estão disponíveis ou publicados na *Web*.

Para verificar a validade dos certificados são utilizadas diversas técnicas como as CRLs para manter uma lista de todos os certificados anulados ou suspensos e um *Lightweight Directory Access Protocol* para conter, em tempo real, os certificados válidos.

Quanto à arquitetura foi utilizada a chamada DigiDoc, que fornece um servidor que opera diretamente com a base de dados dos certificados da CA raiz e providencia confirmações relativamente à validade dos certificados e das assinaturas. Em 2002, o formato XML-DSIG era o mais utilizado, contudo, este só permite uma assinatura por documento. Logo de seguida, passaram a utilizar o formato de documento XAdES que já permite diversas assinaturas por documento (Sertifitseerimiskeskus 2003).

Em Portugal, com o cartão do cidadão podemos começar a desenvolver plataformas deste género, tirando partido destes certificados digitais apropriados para assinaturas digitais. Porém, não existe ainda uma grande aposta na utilização dos mais variados serviços que o cartão do cidadão pode, potencialmente, oferecer.

Muitas das funcionalidades que este cartão tem são também pretendidas nesta dissertação, principalmente o facto de se poder assinar digitalmente e aceder a registos médicos. Vê-se, mais uma vez, que é um elemento considerado seguro, tendo em conta as áreas em que é utilizado.

Em termos de plataformas mesmo concretas existem já diversas ferramentas que permitem fazer assinaturas digitais. Há diversas bibliotecas que fornecem uma *Application Programming Interface* (API) para as mais diversas linguagens de programação e para as mais diversas tecnologias *Web* e *desktop*. Normalmente os *Smart Cards* instalam os seus certificados digitais na *Certificate Store* do sistema operativo que se estiver a utilizar, desde que esteja instalada nessa máquina um *middleware* que instala os *drivers* e todas as formas de acesso ao conteúdo do cartão. Estas ferramentas conseguem ir buscar os certificados às *Certificate Stores* e dessa forma aceder ao certificado digital que se pretende sem grande problema. Por exemplo a biblioteca "SecureBlackBox" utilizada na implementação do Sistema Multiplataforma, fornece uma API com funções de assinatura digital para as linguagens de programação mais conhecidas. Depois existem outros programas que permitem assinaturas digitais básicas para XML, por exemplo. As próprias linguagens Java e C# permitem efetuar, nativamente, assinaturas digitais básicas para XML. Depois existem também programas para efetuar assinaturas digitais em PDF, documentos *MSOffice* e outros. Para cada caso individualmente existem já um ou outro *framework* que permite desenvolver programas à volta a partir das funcionalidades de assinatura que eles contêm.

2.6 Técnicas para assinar digitalmente

Em termos de técnicas criptográficas, que permitam assegurar com toda a confiança que um documento foi assinado digitalmente e não foi alterado desde então, são utilizadas técnicas de chaves públicas. Só desta forma conseguimos garantir que um documento foi assinado por uma entidade em concreto. Num destes casos, por exemplo, o documento é assinado, e pode ser enviado, juntamente, com o certificado digital, da entidade que está a assinar, e este certificado está assinado pela CA. Posteriormente, para aceder ao documento tem que se obter a chave pública da CA, portanto, tem de ser um elemento seguramente certificado pela CA. Depois o certificado é decifrado, obtendo, assim, a chave pública da entidade que assinou, podendo assim decifrar-se o documento, com a confiança que aquele documento é o original. Este é um exemplo de envio de um documento assinado digitalmente. Numa comunidade que confia na mesma CA, as chave públicas uns dos outros são obtidas facilmente.

2.6.1 RSA, algoritmo Rivest Shamir Adleman

Em termos de técnicas que permitam uma assinatura digital existem algoritmos criptográficos de chaves públicas, tais como o RSA (Rivest, Shamir, and Adleman 1978), que é uma técnica imensamente utilizada na *Internet*, em *e-mails*, compras eletrónicas e afins.

As chaves são geradas da seguinte forma:

- Escolhemos de forma aleatória dois números primos p e q (por exemplo da ordem dos 2^{332});

- Obtemos um $n = p * q$ e uma função *totient* $\Phi(n) = (p-1)*(q-1)$;

- Escolhemos um número inteiro e, tal que $1 < e < \Phi(n)$, de forma a que e e $\Phi(n)$ sejam primos entre si;

- Geramos um d de forma a que $e*d = 1 \bmod (\Phi(n))$, utilizando, por exemplo, o algoritmo de Euclides.

- A chave pública será o par de números n e e;

- A chave privada o par de número n e d;

- Para cifrar uma mensagem m, onde $0 < m < n$, numa mensagem cifrada c basta fazermos: $c = m^e \bmod n$;

- Para decifrarmos $m = c^d \bmod n$.

Esta técnica é das melhores técnicas assimétricas e das mais utilizadas, porém é um algoritmo considerado bastante pesado computacionalmente, mesmo que seja efetuado num bom *hardware*. A maior parte das técnicas assimétricas são mais pesadas computacionalmente do que as técnicas simétricas. Porém só com base nelas é que conseguimos obter de uma assinatura digital os conceitos de integridade e autenticidade.

2.6.2 DSA, Digital Signature Algorithm

O *Digital Signature Algorithm* (DSA) foi proposto pelo Instituto Nacional de Standards e Tecnologia em 1991 (Kravitz 1993).

É também uma técnica de chaves públicas e a geração das chaves tem duas fases. A primeira fase envolve escolha de parâmetros, que podem ser partilhados entre diferentes utilizadores do sistema, e a segunda fase envolve gerarmos as chaves públicas e privadas de um só utilizador. Para gerarmos as chaves público-privadas, tendo em conta o conjunto de parâmetros definidos anteriormente, devemos: escolher um número aleatório x, onde $0 < x < q$, sendo q um primo N-bit, tal que N é menor ou igual ao tamanho do *output* da *hash* a ser assinada; calcula-se $y = g^x \bmod p$, sendo que p é um número primo L-bit, tal que p-1 seja um múltiplo de q, e g é um número cuja ordem multiplicativa do módulo de p for q; a chave pública é então (p, q, g, y) e a chave privada é x.

A ideia por trás de todos estes algoritmos é utilizarmos processos matemáticos reversíveis de forma a que tudo o que seja cifrado, possa ser decifrado. As técnicas de chaves públicas são normalmente pesadas computacionalmente, devido precisamente às custosas operações matemáticas, porém são as mais eficazes em termos de segurança.

2.7 Resumo

Em suma, podemos verificar que pegando em pequenas partes que já existem e escolhendo e analisando propriamente cada uma delas, é possível gerar uma infraestrutura de assinaturas digitais útil a esta dissertação. Em termos de gestão de políticas e aplicação de segurança com criptografia existem já diversas formas muito utilizadas e testadas que asseguram integridade a todo o processo.

As infraestruturas de chave pública parecem as mais adequadas, pois permitem o controlo das origens de assinaturas por parte dos médicos, o que assegura maior confiança informática.

Quanto aos *Smart Cards*, estes são utilizados nas mais diversas áreas, sendo que todas elas envolvem segurança e confidencialidade. Vimos como os dados são guardados dentro do cartão e como podem estar protegidos. Tendo em conta as áreas em que estes são utilizados, concluímos o uso destes cartões é um método eficaz de proteger os dados e de autenticar as entidades envolvidas no problema. Quanto aos algoritmos utilizados, devido à matemática envolvida, que faz torna as cifras computacionalmente complexas, o RSA apresenta-se como o mais eficaz e o mais utilizado, o que apresenta uma maior confiança de utilização.

Com base no que vimos que está por trás de todo um contexto que envolve criptografia de chaves públicas, concluímos que aliar o *Smart Card*, individual e único para cada médico e que contém as suas chaves criptográficas, a uma plataforma *Web* que permita ao cliente efetuar assinaturas digitais e que se adapte aos mais diversos sistemas operativos e plataformas,

consegue-se atingir o objetivo esperado, tendo sempre em conta que depende que esteja bem implementado uma CA e tudo o que isso implica.

Capítulo 3

3. Descrição da Tecnologia a usar

3.1 Tecnologia a utilizar

Pretendendo-se configurar um sistema multiplataforma, será necessário usar, à partida, as mais diversas tecnologias. Os casos de utilização que o sistema tem são os representados na figura 2 e estes devem ser possíveis efetuar em qualquer plataforma digital.

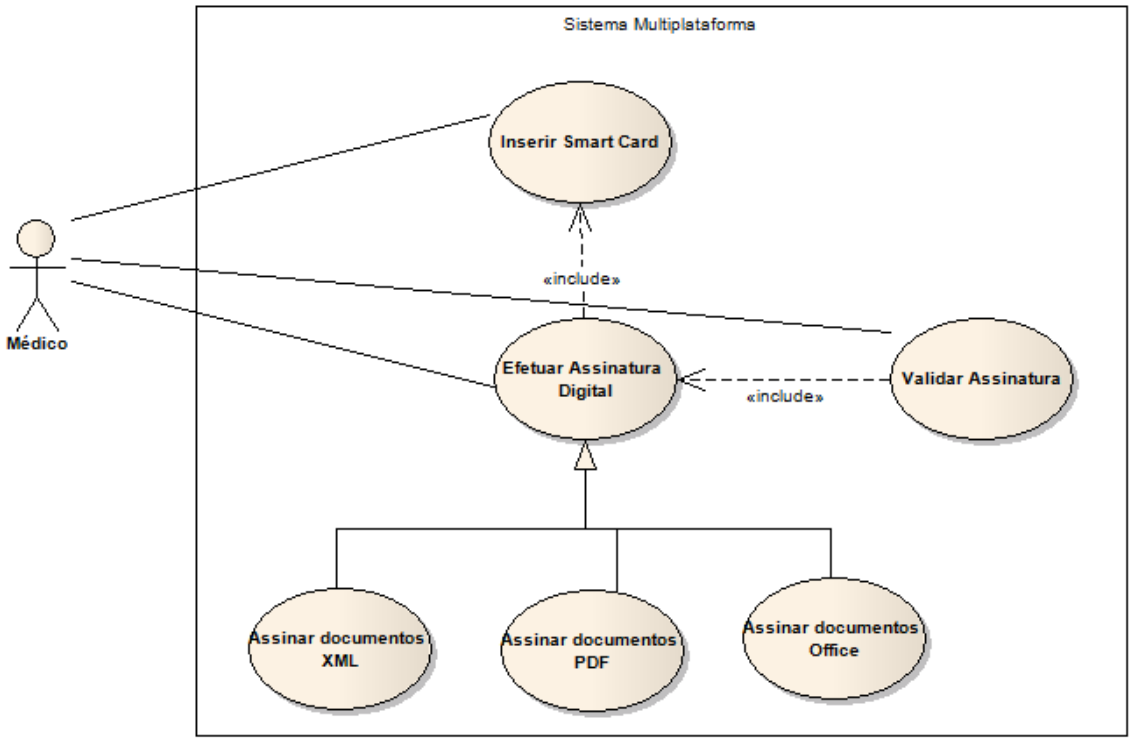

Figura 2: Modelo de casos de utilização

Porém, estas devem ser genéricas, comuns às plataformas de interesse para evitarmos usar múltiplas ferramentas e linguagens de programação. Como ponto base, o sistema a construir tem de conseguir usar os *drivers* do *Smart Card* através de pedidos efetuados a partir dos navegadores *Web* mais conhecidos (*Mozilla Firefox*, *Google Chrome*, *Opera*, *Safari* e *Internet Explorer* pelo menos). Por motivos de segurança não achamos boa ideia que o lado do servidor acedesse aos dados do cartão, pelo que o acesso será efetuado do lado do cliente, como iremos detalhar mais à frente. Decidimos desenvolver uma aplicação *Web*, acessível por qualquer um destes navegadores *Web* referidos. Os exemplos referidos no capítulo da Revisão Bibliográfica referem que o *gateway* que liga o navegador ao conteúdo do cartão é, em princípio desenvolvido em Java, mas para isso é necessário haver uma *Java Virtual Machine* implementada no sistema operativo. A maior parte dos sistemas conseguem tê-la instalada, porém alguns não, principalmente em sistemas operativos móveis. Nesses, teremos de usar uma solução que vai de encontro às suas capacidades. Este caso é excecional, pois a orientação que pretendemos seguir é minimizar o número de adaptações e tornar o sistema realmente independente e preparado para qualquer plataforma existente.

Na figura 3 está uma representação gráfica dos componentes principais, a desenvolver, de todo o sistema. Quanto aos dispositivos, mostram-se apenas três, usados para exemplificar.

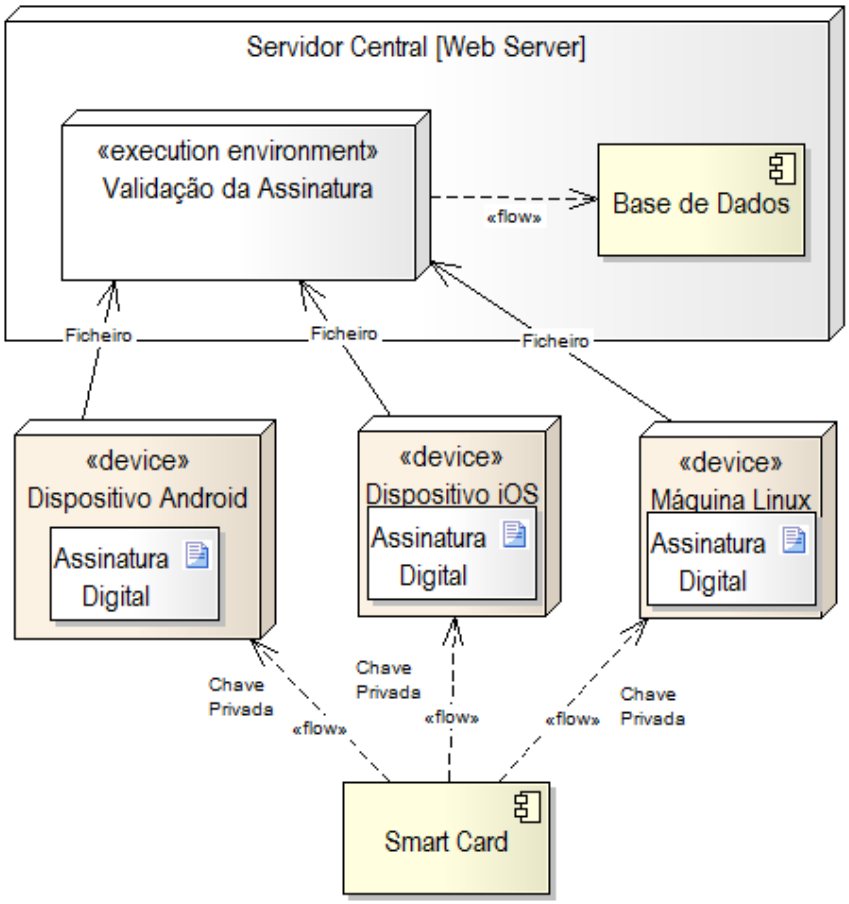

Figura 3: Exemplo Alto-Nível do Sistema Multiplataforma

O Sistema será uniforme e visto para o utilizador como é visto na imagem acima.

O sistema central que que efetua as assinaturas e as validações aos documentos foi desenvolvido com a *framework* .NET (versão 4.5), utilizando a linguagem de programação C# e é uma aplicação *Web* ASP.NET com a arquitetura *Model-View-Controller* (MVC) 4 (Lindberg and Rydin 2001). Estas opções foram escolhidas para o sistema ser mais facilmente integrado no contexto das soluções Glintt Healthcare Solutions. Em relação à base de dados, esta deve ser preferencialmente Oracle (uma vez que é a tecnologia de base de dados mais utilizada na Glintt), caso o cliente final a possua, ou então, em alternativa uma base de dados, por exemplo, como a SQL Server. Contudo, toda a camada de interação com a base de dados opera independentemente da tecnologia, para que possamos adaptá-la ao maior número possível de clientes finais da aplicação.

Desenvolvemos também um Java *applet* para que os sistemas operativos fixos como o *MSWindows*, *Linux* e *Mac OS* e, pudessem usar a partir da *Java Virtual Machine* neles instalada. O *applet* aparece ao utilizador quando este entra no navegador *Web*. Executa no cliente, logo disponibiliza métodos para que as assinaturas digitais sejam efetuadas com recurso ao *Smart Card*. Iremos explicar o porquê de todas as decisões, mais detalhadamente, nos próximos capítulos de arquitetura e implementação do sistema. Para os sistemas operativos móveis foi desenvolvida uma aplicação para *Android* e outra para *iOS* que é chamada a partir do *WebSite* principal. Estas aplicações efetuam a assinatura digital localmente e acedem ao *Smart Card* para ler a chave privada lá contida e devidamente protegida e poder efetuar a assinatura.

Quanto ao *Smart Card* escolhido para efeitos de teste, optamos pelo cartão de cidadão português, que contém dois pares de chaves, no qual um deles é usado exclusivamente para assinaturas digitais. A tecnologia neste cartão, utilizada na maior parte dos *Smart Cards*, é das mais modernas. O *chip* do cartão contém um processador de 8-16 bits, uma *Read-Only Memory* (ROM) responsável pela comunicação, sistema operativo e operações criptográficas, uma *Electrical Erasable Programmable Read-Only Memory* (EEPROM) com o sistema de ficheiros, chaves e PINs de proteção; contém ainda uma componente *Random-Access Memory* (RAM), um gestor de contactos mecânicos e ainda uma parte que trata da segurança física do cartão. O código de comunicação gere tanto a baixo nível, de caracteres ou bloco de bits, como alto nível, com as chamadas APDU (QREN 2007).

3.2 Algoritmos

Foi necessário escolher entre diversos algoritmos de criptografia e também dentro de vários algoritmos de *hashing* os que mais se adequavam ao fim em vista. Tendo em conta que o *Smart Card* que iria ser utilizado como caso de teste foi o cartão do cidadão, que exige assinaturas num único algoritmo específico, o RSA ficou logo selecionado para a encriptação-desencriptação. Os certificados digitais contidos no cartão do cidadão português só podem ser

utilizados com o algoritmo RSA. O cartão exige também que o algoritmo SHA-1, para cálculo de *digests* dos documentos.

Quanto às bibliotecas de assinatura digital, foram estudadas pelo menos três. A "SecureBlackBox" (Eldos 2014), a "iText" (iText 2014) e a "XAdES.NET project" (Palomino and Palomino 2014). Foram efetuados pequenos testes de cada uma delas, com base na documentação respetiva.

No caso de documentos XML foram testadas duas bibliotecas, como foi referido no capítulo da Implementação. Em termos de rapidez o "XAdES.NET project" demora em média 423 milissegundos, sendo que foi testado trinta vezes com o mesmo documento XML. Este documento é um documento médio com 6,34 quilobytes. Já o "SecureBlackBox" demora, para os mesmos testes, 507 milissegundos em média. É ligeiramente mais lento, porém é mais completo e fornece mais algumas opções de assinatura digital, na *tag* de assinatura do XML, sendo fornecidas muito mais informações e muito mais detalhe sobre os métodos e os algoritmos utilizados na assinatura. Tendo em conta que a licença desta biblioteca é consideravelmente cara, o "XAdES.NET project" foi considerado uma boa alternativa, tendo em conta que é gratuita.

De seguida, partimos para testes em documentos PDF. Neste caso as duas bibliotecas testadas foram novamente a "SecureBlackBox" e o "iText" que possibilita assinaturas em documentos PDF. Em termos de desempenho, pode-se concluir que são muito semelhantes, apresentando um tempo de execução de poucos milissegundos. Com um documento PDF de quarenta páginas, com bastantes textos e imagens (sendo que é um jornal digitalizado) no "iText" a assinatura demorou 1337 milissegundos em média. Esta média teve trinta amostras, tendo sido executado o programa trinta vezes. Na "SecureBlackBox" no mesmo documento a assinatura demora em média 490 milissegundos. É consideravelmente mais rápido, o que pode ser um fator a considerar, devido ao número elevado de assinaturas que irá haver. A "SecureBlackBox" tem ainda mais opções para se detalhar mais campos que a assinatura digital pode conter. Porém, é uma API com um custo elevado de licença, enquanto que a do "iText" é completamente gratuita. As assinaturas foram efetuadas nas mais diversas plataformas novamente, e todas efetuaram a assinatura com sucesso.

Quanto à assinatura digital dos documentos do *MSOffice* a única biblioteca encontrada foi a do "SecureBlackBox". Esta permite assinaturas para os mais variados documentos.

Na tabela 1 vemos uma comparação entre as bibliotecas encontradas e testadas.

Tabela 1: Comparação entre bibliotecas de assinaturas digitais

	SecureBlackBox	XAdES.NET	íText
Assinaturas XML	✓	✓	-
Assinaturas PDF	✓	-	✓
Assinaturas Office	✓	-	-
C#	✓	✓	✓
Java	✓	-	✓
Smart Card	✓	✓	✓
Validação XML	✓	✓	-
Validação PDF	✓	-	✓
Validação Office	✓	-	-
Rapidez	Grande	Grande	Grande
Custo	Elevado	Grátis	Grátis
Documentação	Completa	Pouca	Razoável
Suporte	Completo	Pouco	Razoável

Por uma questão de uniformização do código e utilização de apenas uma biblioteca, foi escolhida a "SecureBlackBox", apesar de ter um custo de licença.

3.3 Resumo

Em suma, pretendemos que a tecnologia utilizada estivesse disponível para plataformas de interesse. Tanto a nível de base de dados como de assinaturas digitais, a tecnologia utilizada está preparada para funcionar corretamente com qualquer tipo de plataforma, abrangendo assim grande número de dispositivos e de sistemas operativos. O Sistema Central foi desenvolvido em .NET C#, para que o integrássemos facilmente nas soluções da empresa Glintt. No resto, usamos também tecnologias genéricas, exceto nalguns casos individuais.

Capítulo 4

4. Arquitetura do Sistema

4.1 Problemática da utilização da chave privada

Era um requisito central do trabalho servir a todos os clientes *Web*, para que pudesse ser utilizado pelos mais diversos sistemas operativos e plataformas. A ideia principal era ter um *WebSite* que contivesse serviços de assinatura e validação de documentos recebidos dos clientes.

Acontece que para haver um serviço *Web* que possa assinar documentos em nome de um utilizador, este tem de ter conhecimento da chave privada desse utilizador. O facto de o *WebSite* (lado do servidor) ter conhecimento da chave privada do cliente é um mau princípio, que viola o conceito de chave privada. Para começar, a chave privada teria de ser enviada (juntamente com o certificado) para o servidor, o que implica que a chave privada circule na rede. Contudo, este envio podia ser feito por um canal SSL/TLS de modo a que os dados estivessem protegidos de terceiros. Ainda assim, é violado o princípio de que a chave privada é apenas do conhecimento da entidade que a possui e deve ser utilizada apenas localmente. A alternativa a esta situação é desenvolver uma aplicação nativa para cada sistema operativo, tentando utilizar tecnologias de programação o mais abrangentes possível, de modo a generalizar o mais possível e ter uma arquitetura de sistema globalizada.

Uma outra alternativa estudada, foi o médico necessitar de se autenticar devidamente, elaborar e assinar o documento localmente e, de seguida, enviar o documento ao *WebServer* principal. Este verificava a integridade do documento e assinava-o também, ficando o documento com duas assinaturas. Ponderámos esta hipótese por possivelmente dar mais integridade ao processo. Duas assinaturas poderiam ser melhores do que uma. A ideia neste caso é autenticar primeiro o médico e estando lá a assinatura do servidor central era sinal que a assinatura era validada previamente antes de ser guardada na base de dados. Porém, o foco do

sistema não é a instituição, mas sim o médico. A partir do momento em que o médico assina o documento, sabe-se que foi aquele médico a assinar e é essa garantia que se pretende.

Pelo exposto, começámos por desenvolver os algoritmos de assinatura, mas desta vez na linguagem Java. A ideia era correr um *applet* de Java a partir do lado do cliente, chamado a partir do código HTML. Testámos e conseguimos chamar o *applet* e as suas funcionalidades de assinatura, assegurando assim que a chave privada não saia para o exterior do lado do cliente. Depois das experiências em *MSWindows*, *Linux* e *Mac OS*, passámos aos sistemas operativos móveis como o *iOS*, *Android*, e *Windows Phone*. Logo no *iOS* que é o mais prioritário, percebemos que esta solução não iria ter sucesso, pois este sistema operativo não consegue ler a linguagem Java, impossibilitando o uso de *applets* de Java. Mesmo sendo o código HTML ou *javascript* a ler e a chamar os *applets*, se não houver uma *Java Virtual Machine* instalada na máquina que está a aceder ao navegador *Web*, não se consegue aceder aos métodos contidos nesse mesmo *applet*.

Suspendemos, por momentos, a abordagem do Java *applet*, tendo a abordagem seguinte sido o uso maciço de *javascript*. Estudamos, na subsecção seguinte, mais a fundo se valeria a pena desenvolver nativamente em *javascript* rotinas e métodos de criptografia.

4.1.1 Criptografia com JavaScript

Toda esta subsecção foi baseada em (Matasano 2014).

Para tentar chegar a todos os clientes com apenas uma linguagem de programação pensou-se desenvolver então uma biblioteca de origem de assinaturas digitais em *javascript*, uma constante em todos navegadores *Web* modernos. Ora, se não se confia num servidor para enviar uma senha ou uma chave privada, não se pode também, à partida, confiar nele para nos disponibilizar código-fonte seguro. O código *javascript* é descarregado desde o servidor até ao cliente, logo está sujeito a alterações. O código-fonte do servidor pode ser alterado, se este estiver comprometido, e todo o *javascript* disponibilizado pode ser alterado, o que faria com que as rotinas de assinaturas digitais fossem facilmente manipuladas. Porém, podemos transmitir o código através de um canal SSL/TLS, mas, nesse caso, não é necessário *javascript*, porque já temos criptografia a sério estabelecida, logo não compensaria. Neste caso, mais valia desenvolver em qualquer outra linguagem e descarregar o programa pelo canal seguro. Para além disto, para transmitir não basta simplesmente enviar o ficheiro *javascript*, mas sim todo o conteúdo da página *Web* através do canal seguro. Caso contrário, os atacantes irão obter o código de criptografia, utilizando o ponto de ligação menos seguro, que existir, na altura em que a página é construída.

Também um fator de risco associado a esta linguagem é o facto que não conseguimos focar num só pedaço de código *javascript* sem ter de avaliar todas os outros pedaços de conteúdo que ajudam a construir toda a página *Web*, pois o pedaço de *javascript* é apenas um componente de uma página *Web*. A rotina de criptografia no *javascript* pode ser completamente insegura, pois

basta estar inserido numa página que tenha um atributo invisível que altere outras rotinas da qual a rotina de criptografia depende, utilizando, por exemplo, um *backdoor*. Para resolver isto bastava calcular um *digest* do código-fonte do servidor para que este se pudesse verificar a si próprio, aquando o código fosse descarregado para o cliente, comparando esse *digest* do servidor com o seu próprio *digest* naquele momento. Acontece que isto não funciona pela razão já explicada, ou seja, não pode ser só o *javascript* a ser verificado, porque é necessário avaliar outros pedaços de conteúdo que ajudem a construir toda a página *Web*. Estas vulnerabilidades são bastante exploradas nos ataques informáticos conhecidos como "*cross-site-scripting*".

Outro fator de insegurança e de risco é a maleabilidade do contexto de execução do *javascript*, o que quer dizer que é possível alterar o modo como o ambiente funciona em tempo real. Praticamente qualquer função na qual a criptografia dependa, pode ser reescrita silenciosamente por qualquer outro pedaço de conteúdo utilizado para construir a página. Não há nenhuma forma confiável para que qualquer pedaço de *javascript* verifique o seu ambiente de execução. Por exemplo, o *javascript* não pode perguntar se está a lidar com um número aleatório que foi fornecido por um atacante. Não há forma de distinguir se é um número fornecido de uma entidade fidedigna ou não. Não pode também garantir que ninguém pode fazer algo com o segredo criptográfico, exceto em alturas que o autor o aprove. E estas duas garantias são propriedades vitais para um correto funcionamento criptográfico, mas devido ao facto do *javascript* ser *runtime*, estas não podem ser garantidas. Contudo, poder-se-ia escrever uma simples extensão do navegador que permitisse ao *javascript* verificar-se a ele próprio, mas é bastante trabalhoso, porque teríamos de verificar todo o *runtime*, com a ajuda do *Document Object Model* do *javascript*.

Uma das funcionalidades de programação que o *javascript* não contém é gerar um número aleatório de uma forma segura, o que é grave, pois toda a criptografia depende da segura geração de números aleatórios. Obviamente que o *javascript* poderia obter números aleatórios fidedignos através da WWW e utilizá-los para efetuar as operações criptográficas, mas fazê-lo sem um comunicação com um canal SSL/TLS, ou um outro tipo de canal seguro, era um risco novamente, e para utilizar este tipo de canal porque precisamos de criptografia com *javascript*? Voltaria a não compensar, pois voltaríamos a ter já criptografia a sério.

Um fator de risco associado é ainda o facto de esta linguagem só fazer *garbage collection* periodicamente, o que pode implicar que os segredos que estão na memória podem ser descobertos, por estarem lá mais tempo do que aquilo que realmente necessitariam de estar.

Em suma, aliado ao facto das operações criptográficas em *javascript* não estarem muito desenvolvidas, não é muito seguro desenvolver operações de assinaturas digitais, pois estas podem estar sempre a ser manipuladas e alteradas por outra entidade maliciosa. Logo, a opção mais generalizada possível, de forma a ir ao encontro do sistema multiplataforma, é utilizar então um Java *applet* para ser lido do lado do cliente. A linguagem Java consegue ser utilizada em ambientes *MSWindows Desktop*, *Linux* e *MacOS*. Estas três plataformas irão conseguir ler diretamente o *applet*, pois todos suportam uma *Java Virtual Machine* que interpreta e lê a

linguagem. Fica, portanto, assegurado pelo menos para estes três sistemas operativos, que as rotinas de assinaturas digitais ficam devidamente implementadas de forma igual para todos. Para os sistemas operativos móveis iremos ver na secção seguinte a abordagem escolhida.

4.2 Arquitetura do Sistema Multiplataforma

O Sistema Multiplataforma contém um *Website*, acessível por qualquer dispositivo cliente HTTP. Este sistema é do tipo cliente-servidor.

Sendo para sistemas móveis ou fixos, o médico tem esta interação com o sistema, representada na figura 4.

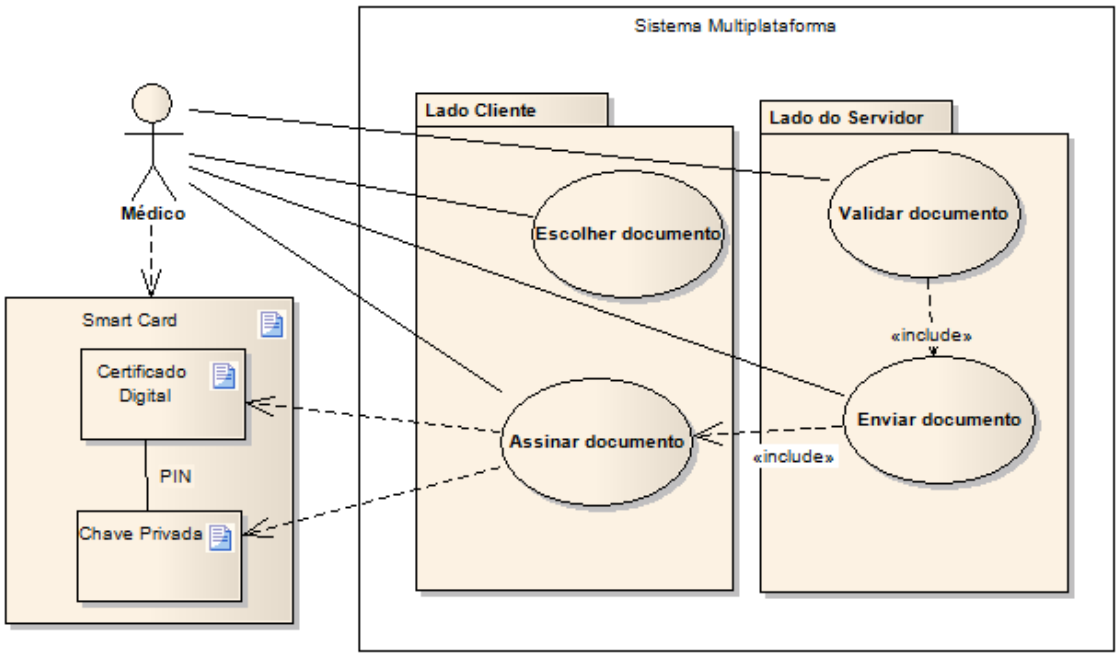

Figura 4: Interação utilizador - sistema

Por um lado existe o componente que diz respeito ao cliente, onde está contido todo o lado visual apresentado ao utilizador. É neste componente que o utilizador recebe toda a informação e interage com o sistema, fornecendo alguns dados como *inputs*, por exemplo. É também nesse componente que desenvolve toda a componente de assinaturas digitais. Tendo em conta todas as restrições estudadas ao longo da pesquisa, optou-se então por desenvolver um *applet* de Java, o que constitui uma boa solução para os sistemas *Desktop MSWindows*, *Linux* e *MacOS*. O código-fonte Java não é descarregado a partir do navegador *Web*, é apenas mencionado uma *tag* no código HTTP com o ficheiro da biblioteca Java. O código em si não pode ser alterado, porque não é descarregado para o navegador, mas sim apenas a referência ao nome da biblioteca Java.

Estando os ambientes *desktop* com as assinaturas funcionais, faltava planear a arquitetura para sistemas operativos móveis. Nestes casos, foi necessário desenvolvermos e instalarmos uma aplicação nativa, apropriada para cada sistema operativo em particular. A hipótese inicial a ser testada foi: quando o dispositivo móvel aceder à aplicação *Web*, através de um navegador, a partir da deteção do tipo de sistema operativo, seja efetuado um pedido interno à aplicação nativa, para que a assinatura seja efetuada. Desta forma, mantemos tudo numa aplicação *Web*.

4.2.1 Plataformas Móveis

No caso do *Android* a ideia foi fazer uma chamada de iniciação da aplicação através de um URL *schema*.

O objetivo é maximizar a interação do cliente com funcionalidades na aplicação *Web* e, desta forma, tal é possível. Não há, portanto, nenhuma interação que o utilizador tenha que efetuar com a aplicação móvel, pois é tudo efetuado automaticamente através do pedido do URL *schema*, como iremos ver no capítulo seguinte da implementação. Esta serve apenas de intermediário para efetuar a assinatura digital localmente e, de seguida, enviar o ficheiro assinado para o servidor da aplicação *Web*.

No caso do *iOS* a abordagem foi semelhante à do sistema operativo *Android*, novamente dando ênfase à aplicação *Web*, e recorrendo à aplicação nativa apenas para efetuar as operações de assinatura digital, implicando assim que o utilizador apenas interaja com o sistema *Web*. Apesar de em *iOS* ser necessário utilizar *Objective*-C para desenvolver aplicações nativas, conseguimos desenvolver esta componente também em Java, com recurso a exatamente as mesmas bibliotecas de assinatura digital, tendo para isso utilizado uma *framework* que o converte, apropriadamente, para *iOS*.

Resumidamente, para utilização do sistema em plataformas *Desktop*, tal como, *Windows*, sistemas com base em *Linux* e *MacOS* o navegador *Web* mostra um Java *applet*, no qual o utilizador tem de aceitar utilizar a aplicação (regra de segurança do Java, que para todos os *applets* pergunta sempre se o utilizador realmente confia no servidor e se quer utilizar a

aplicação. O *applet* tem ainda de estar assinado digitalmente por um certificado digital válido). Quanto a aplicações para sistemas operativos móveis, o utilizador acede na mesma através de um navegador *Web*, porém quando efetua operações de assinatura digital, uma aplicação nativa é lançada, efetuando a assinatura e devolvendo o documento assinado, sem qualquer interação com o utilizador. O utilizador só interage com a aplicação *Web*. Conseguimos assim desenhar uma arquitetura do lado do cliente que não viola nenhum princípio de segurança, mas que é multiplataforma. Todos os clientes utilizam a mesma base de código-fonte de uma biblioteca de assinaturas digitais desenvolvida em Java.

4.2.2 Servidor

Quanto ao lado do servidor, este fora desenvolvido em ASP.NET MVC 4.0, utilizando a linguagem de programação C#, utilizando a *framework* do .NET 4.5.

Neste tipo de arquitetura (MVC) os modelos são classes que representam os dados da aplicação e que utilizam estratégias de validação que asseguram as regras de negócio para esses mesmos dados. As vistas representam os ficheiros *template* que a aplicação utiliza para gerar respostas HTML dinamicamente. As vistas nunca se preocupam com a lógica de negócio nem nunca interagem com a base de dados diretamente; trabalham apenas com dados provenientes dos controladores. Por último, os controladores são classes que tratam os pedidos do navegador *Web*, obtêm os dados do modelo e, de seguida, especificam os *templates* das vistas que devolvem as respostas ao navegador. É considerada uma arquitetura bastante ágil e é muito utilizada na empresa, daí optarmos pela sua utilização.

O lado do servidor do sistema multiplataforma oferece então toda a *interface* do sistema, através das vistas que são maioritariamente baseadas em código HTML. Caso seja detetado que o cliente é um tipo de plataforma *desktop*, a *interface* mostrada é a do Java *applet*, que se encarrega de pedir ao utilizador para selecionar o ficheiro a ser assinado digitalmente, e assina-o de seguida, e envia-o ao servidor. Caso seja um tipo de sistema móvel, temos a possibilidade de selecionar um ficheiro local e pedir para este ser assinado através de HTML normal. Posteriormente, os controladores analisam o tipo de ficheiro e constroem um URL apropriado para enviar à aplicação móvel a informação correta para este assinar o documento e enviá-lo, já assinado, para o servidor.

Basicamente o servidor é um agregador do lado do cliente e do lado do servidor. É todo o sistema em si que se encarrega de dividir os diversos componentes e utilizá-los nas devidas alturas.

Na figura 5 pode-se ver um diagrama retratando toda a interação dos mais diversos componentes do sistema multiplataforma.

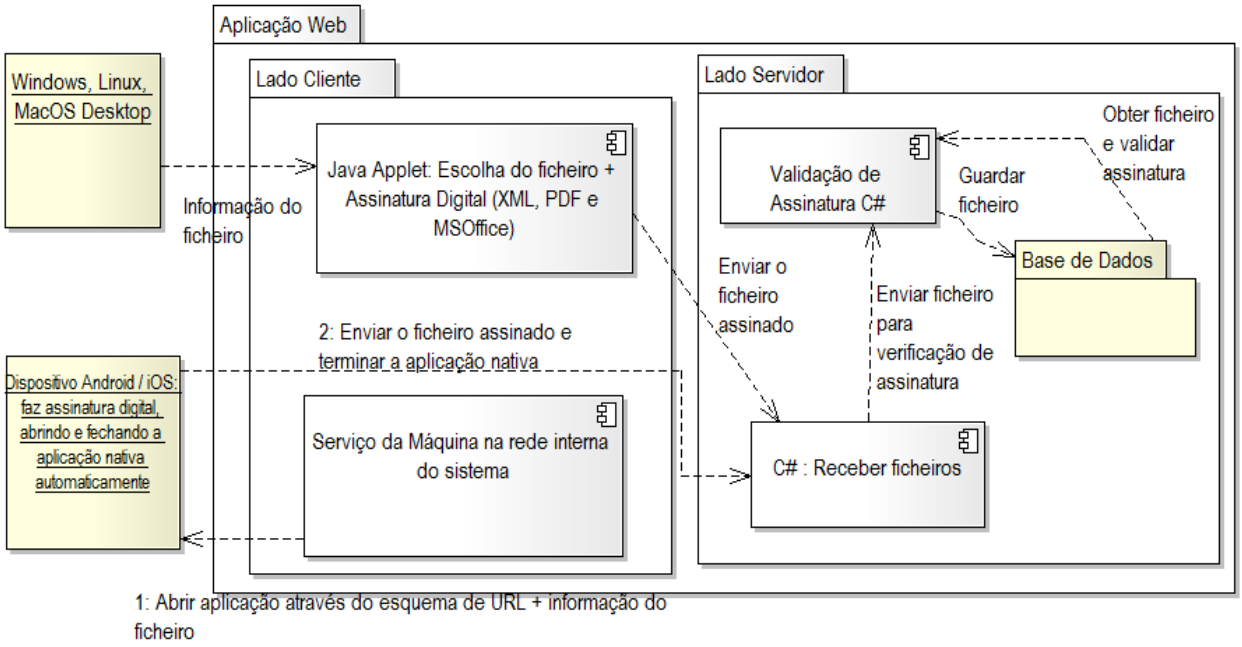

Figura 5: Interação entre os componentes do sistema multiplataforma

4.3 Smart Card

Quanto à leitura de *Smart Cards*, no caso o cartão de cidadão português, para ambientes *Desktop*, os dados irão ser lidos pelo Java *applet*, pois há uma API disponibilizada para algumas linguagens e uma delas é a linguagem Java. Para além da API, é necessário que a máquina tenha instalado o *middleware* do cartão que trata da utilização dos *drivers* apropriados e faz a ligação ao cartão com chamadas com comandos APDU, como vimos no capítulo da Revisão Bibliográfica. Este *middleware* é disponibilizado pelos produtores do cartão de cidadão, uma vez que só eles conhecem detalhadamente a arquitetura do cartão e do seu *chip*. Até por questões de segurança, a informação de controlo, a nível de ligação de *hardware*, não é disponibilizada para o público geral, caso contrário qualquer pessoa poderia manipular, programaticamente, da forma que quisesse o conteúdo do cartão. É, por isso, um sistema fechado. O *middleware* está disponível para transferência e instalação para ambientes *Windows*, sistemas *Linux* e para ambientes *MacOS*. Para todos eles existe uma versão para arquiteturas com 32 ou 64 bits.

Por efeitos de restrição em relação aos tipos de *middleware* disponibilizados para plataformas móveis pelos produtores do cartão do cidadão português, as aplicações nativas não serão integradas na aplicação final. Sem este *middleware* não se consegue utilizar a chave privada que está protegida por PIN no interior do cartão do cidadão. O dispositivo sem este elemento não sabe como aceder ao cartão, nem que comandos APDU deve utilizar para

comunicar com o chip no interior do cartão. As aplicações nativas, como vimos, estão prontas a poder efetuar assinaturas digitais, porém não se consegue integrar o *Smart Card* de teste com este tipo de plataformas ainda. A solução adotada para este tipo de sistemas operativos está mais detalhada no capítulo *Smart Card*.

O cartão do cidadão tem dois pares de chaves público-privadas, um apropriado apenas para autenticação e outro para ser utilizado em assinaturas digitais.

Devido à não disponibilização de um *middleware* para sistemas operativos móveis como o *Android* e o *iOS*, optámos por uma outra abordagem, para que estes sistemas não deixem de poder efetuar assinaturas digitais. Prevê-se que o *middleware* para plataformas móveis esteja disponível daqui por algum tempo, porém ainda não há grande aposta nesta área, pois não há ainda um grande desenvolvimento de leitores de cartões.

Quer-se na mesma que haja uma solução para dispositivos móveis que não tenham um *middleware* desenvolvido, como neste caso do cartão do cidadão. Enquanto não se desenvolve este componente, a ideia foi que o utilizador submeta um código numérico na aplicação *Web*, juntamente com o pedido de assinatura, sendo que esse código está associado a uma máquina na rede interna do servidor. O médico insere então esse código que corresponde a uma máquina em particular, e após isso, vai a essa específica máquina e insere lá o seu *Smart Card*, sendo a assinatura efetuada nessa mesma máquina. O documento que foi pedido para ser assinado é enviado para essa máquina, isto é, para o lado do servidor e o médico insere o cartão nessa mesma máquina. Essa máquina acaba por fazer de cliente e de servidor, pois o lado do servidor vai ao lado do cliente dessa máquina buscar a informação necessária para assinar digitalmente o documento. A chave nunca chega a ser utilizada na rede, mas sim numa máquina (no lado do cliente) que está na rede. Essa máquina tem o leitor de cartões ligado e o médico insere lá o seu cartão. E desta forma não se está a enviar a chave privada criptográfica do médico para a rede, porque como esta vai ser utilizada na mesma máquina, o lado do servidor e o lado do cliente são partilhados pela mesma máquina. Resumidamente, no dispositivo móvel o médico insere um código numérico que corresponde a uma certa máquina e, de seguida, vai a essa mesma máquina e insere lá o seu *Smart Card* que permite que essa máquina utilize a sua chave privada para assinar o documento. O mais importante é que se garante que haja, na mesma, segurança e não se envie a chave privada para a rede. Mas garante-se também que o sistema é um sistema multiplataforma. A figura 6 demonstra como fica o sistema com esta abordagem.

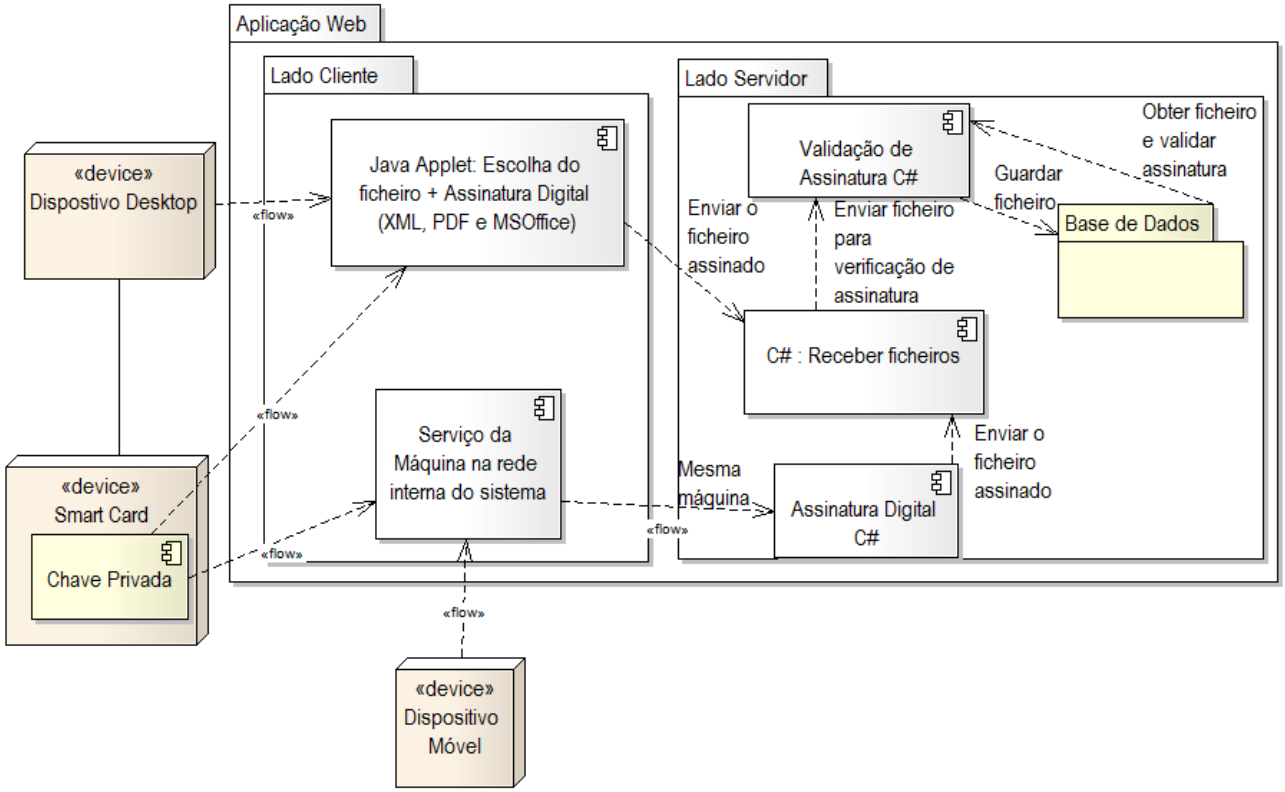

Figura 6: Modelo de toda a interação entre os componentes com abordagem alternativa a *Smart Cards* sem *middleware* para sistemas móveis

Naturalmente que neste caso os sistemas móveis deixam de estar realmente móveis por estarem dependentes de uma outra máquina fixa que permita a inserção do cartão, contudo dá na mesma conforto ao médico em termos de utilização da plataforma que está mais habituado utilizar. Ainda assim, achámos importante arranjar uma solução provisória para *Smart Cards* que não tivessem ainda *middleware* desenvolvido para sistemas operativos móveis.

4.4 Resumo

Em suma, a arquitetura do sistema sofreu diversas alterações motivadas por restrições operacionais, para todo e qualquer tipo de plataforma, e seguro. O grande objetivo era convergir todos os conceitos de segurança e de criptografia que envolvem uma assinatura digital num sistema que operasse sobre os mais variados sistemas operativos. Após várias alterações, ou por restrições tecnológicas, ou por motivos de segurança, conseguiu-se chegar a um sistema centralizado, seguro e multiplataforma como era desejado.

Concluímos que para conjugar um sistema que trabalhe com criptografia, tentando garantir integridade e confiança de processos, com algo multiplataforma, é necessário diversos cuidados na abordagem e na forma como tudo fica organizado e desenhado. A hipótese levantada

inicialmente foi concretizada, porém teve de haver algumas secções divididas, como a parte dos sistemas operativos fixos e móveis, onde a abordagem para cada um destes tipos teve de ser diferente.

Capítulo 5

5. Implementação

5.1 Assinatura Digital e Respetiva Verificação

Vamos iniciar o detalhe da implementação no tópico mais nuclear da dissertação. Iremos ver como foi implementada a assinatura digital e como esta é verificada. Uma assinatura digital sem poder ser verificada, vale de muito pouco, pois o objetivo é mesmo que se verifique se o ficheiro é ou não íntegro. Pretendeu-se efetuar assinaturas digitais de ficheiros com a extensão XML, *Portable Document Format* (PDF) e *MSWord* uma vez que estes são os tipos de documentos que os médicos vão trabalhar no seu dia-a-dia (Glintt 2014a). Para o fazer utilizámos uma sintaxe XML como o XAdES-T que, para além de assinar, adiciona um campo com um *timestamp* para certificar também a data de assinatura do documento.

Pretendemos criar uma *Dynamic-Link Library* (DLL) em .NET para ser utilizada pelo programa final na verificação e validação das assinaturas digitais. Esta DLL será utilizada todas as vezes que for submetido um ficheiro novo. Antes de o guardar na base de dados, o ficheiro deve ser validado para se ter a certeza que a assinatura digital associada é íntegra.

A biblioteca de assinaturas digitais, no formato *Java ARchive* (JAR), está desenvolvida em Java, onde foi criada um *applet* que contém os métodos de assinatura, andar com o cartão do cidadão. Os médicos utilizam-na quando quiserem assinar digitalmente um documento com a sua chave secreta / privada.

Uma assinatura digital permite verificar uma possível adulteração indevida de um documento e também associar o documento ao seu autor. Assim, o leitor do documento sabe de certeza quem foi o escritor e este não pode negar que foi ele que o criou. Para além disto ninguém, mesmo o leitor, pode alterar o documento original sem que tal seja detetado.

Para efetuar assinaturas digitais há duas técnicas. Uma é a de chaves públicas, onde o documento é cifrado com a chave privada do seu autor e o leitor, posteriormente, decifra os dados com a chave pública do autor (já do seu conhecimento). Acontece que esta técnica pode

ser bastante ineficiente se o documento a assinar for muito longo. Os algoritmos de chaves públicas, como vimos no capítulo de revisão bibliográfica, são pesados computacionalmente, e no caso do documento ser longo irá haver alguma demora durante o processo. Outra técnica é a de criar um sumário (*digest*). Consiste em calcular um sumário / *digest* do documento, num dado formato, calculando uma *hash*. Existem diversas técnicas de *hashing* que consiste em transformar uma grande quantidade de dados em uma pequena quantidade de dados. É um método que comprime os dados de tal forma que o resultado gerado é único. Também é suposto que esta informação gerada seja irreversível, ou seja, a partir dela não se consiga descobrir a informação original. De seguida, o autor do documento cifra o *digest* com a sua chave privada. Esta cifra é enviada junta com o documento para o leitor. Para o leitor assegurar que aquele documento é fidedigno, deve também calcular uma *digest* do documento e de seguida decifrar, com a chave pública do escritor, o *digest* enviado juntamente com o documento. Após isso compara os *digests* e se forem iguais é sinal que o documento não foi alterado e pode ser utilizado com confiança. Esta segunda técnica é mais vantajosa, pois calcular uma *hash* e encriptá-la é de longe mais eficiente que encriptar um documento inteiro. Por esta razão, optamos por escolher a técnica do *digest* com chaves públicas. Mesmo que as assinaturas sejam efetuadas do lado do cliente, pretende-se que estas sejam efetuadas o mais rapidamente possível, para que o médico não perca tempo com este procedimento e se possa concentrar no que é realmente importante para ele.

Para implementar todo este conceito, foram estudadas algumas bibliotecas de segurança, nomeadamente a, já referida, "SecureBlackBox", que providencia uma API com métodos de assinatura criptográfica em ficheiros XML, PDF e *MSOffice*. No capítulo da Descrição da Tecnologia a utilizar, já se apresentou a razão da escolha da "SecureBlackBox".

Para integrar novos tipos de documentos na aplicação será necessária a pesquisa de novas bibliotecas que lidem com os componentes internos da extensão do ficheiro em causa. A arquitetura do sistema está preparada para detetar novas extensões de ficheiros e criar funcionalidades de assinatura e validação para essa mesma extensão. A biblioteca deve fornecer uma API que possa ser utilizada na linguagem Java, de forma a integrar com todas as plataformas.

5.1.1 Assinaturas em XML

Para efetuar uma assinatura digital em XML é necessário pegar num documento com uma estrutura válida XML, isto é, onde o interior do ficheiro está devidamente construído com *tags* dentro de *tags*, abertos e fechados apropriadamente construindo um esquema válido, com atributos de elementos válidos e com o resto da sintaxe correta em relação ao esquema XML que define esse ficheiro. De seguida, definimos o nó XML que se quer assinar, no caso, a raiz é o nó mais vezes selecionado, para que todo o documento fique assinado. A raiz é o primeiro nó do documento, ou seja é a primeira etiqueta / *tag*, contendo, possivelmente, dentro dela mais

etiquetas. Após a definição do nó devemos escolher o método de *digest* a usar. De seguida, definimos as propriedades de assinatura digital como: o tipo de assinatura que decide se guardamos a assinatura juntamente com o documento ou se guardamos separadamente; o método de *digest* que pode ser o algoritmo MD5, SHA1 ou até SHA 512 bits, etc; o algoritmo de assinatura, neste caso o selecionado foi o RSA, e escolhemos também o certificado digital para efetuar a assinatura, encriptando o *digest* calculado previamente e guardando também a informação da entidade que está a assinar o documento. Posto isto, basta mesmo só calcular o *digest* e de seguida efetuar a assinatura digital. O documento que contém a assinatura é um documento diferente do original, neste caso, e contém toda a informação original concatenada com um novo nó de assinatura, sendo que este está ligado à raiz do documento. Este nó contém o método de assinatura, o valor do *digest*, a *hash* encriptada, ou seja, o valor da assinatura, o nome da entidade que assina, juntamente com alguma informação do certificado e ainda um *timestamp* com a data e hora do momento em que a assinatura digital foi efetuada.

Devido à estrutura da biblioteca, ao carregarmos o certificado digital para a memória, carregamos tanto a chave pública e a chave privada, sendo que para aceder à chave privada temos de introduzir uma senha de acesso. Creio que por uma questão de fácil armazenamento e associação do par de chaves a maior parte dos certificados digitais têm a chave privada como um campo desse mesmo certificado, porém não devia ser assim, pois o certificado digital deve ser do conhecimento de todos e por esse motivo não devia ter um campo, mesmo que protegido, com a chave privada. É uma questão de violação do princípio da confiança. Ao desenvolver a plataforma estamos a associar a chave privada ao certificado, mas apenas porque a *framework* .NET e a biblioteca assim o exigem. É uma questão de programação, onde os produtores juntaram tudo na mesma estrutura de dados, ainda que, teoricamente, sejam coisas separadas. O certificado digital não vem nunca com a chave privada. No caso, na estrutura de dados, a chave privada é um campo do certificado, porém está protegida e só é acessível a quem tem conhecimento da senha / PIN. Trata-se de um certificado digital especial.

Uma vez utilizado o par de chaves de um certificado, é possível verificar se o documento foi ou não alterado. A assinatura sobre o *digest* da mensagem é efetuado com a chave privada, logo a validação da assinatura tem de ser verificada com a chave pública. Se por algum motivo, não se conseguir verificar, é porque a assinatura foi violada, logo o documento assinado deixa de ser considerado válido.

Após a assinatura há que verificar se algum conteúdo no ficheiro XML foi alterado. Esta verificação tem de ser efetuada, porque num documento assinado digitalmente temos de verificar alguma possível alteração. Para verificar este aspeto temos de carregar as referências e as propriedades definidas no documento assinado e validar uma a uma, para testar se houve alguma alteração feita desde o momento da assinatura.

Acabado o processo de assinatura digital no XML, tratámos de desenvolver a assinatura em documentos com o formato PDF.

5.1.2 Assinaturas em PDF

Para documentos do tipo PDF, utilizámos as bibliotecas Java (JARs) necessários que a API "SecureBlackBox" fornecesse e que permitem efetuar operações em documentos PDFs. Esta biblioteca providencia diversos JARs para a linguagem Java e DLLs para a linguagem C# para a validação das assinaturas, por exemplo. A lógica do código para ambas é semelhante, porém está aplicada e adaptada a cada uma delas nos pormenores mais específicos de sintaxe e semântica da linguagem.

Para se efetuar então a assinatura digital, primeiro carregámos o documento para uma estrutura de dados que lida bem com o formato PDF. Segundo, criámos um *handler* de segurança que especifica o tipo de assinatura a utilizar, que, neste caso, o algoritmo de assinatura RSA, sobre certificados X.509, com o algoritmo de *hashing* SHA-1 e também foi definido o tempo e hora de assinatura para que o documento possa ter associado à assinatura um *timestamp*. Com o *handler*, podemos escolher também se a assinatura fica, ou não, visível no documento assinado. De acordo com as preferências da empresa e também para manter a correta visibilidade do documento, mesmo que assinado, decidiu-se que a assinatura iria ficar invisível. Após estas definições a API utilizada calcula o *digest* do texto do documento e procede à assinatura com a chave privada do certificado de teste.

Para efetuar a validação do documento PDF é necessário carregar o documento assinado para a mesma estrutura de dados mencionada acima que suporta e trata os dados em formato PDF. Antes de validar a assinatura, a API obriga a validar o certificado, no sentido de ver se o certificado é válido segundo alguma CA. Se não tiver a assinatura de uma CA válida e autêntica, a assinatura do PDF não é considerada válida. Para validar um certificado digital tem de se verificar toda a cadeia de certificados, sob o formato de árvore. Por exemplo, no certificado digital de assinatura digital no cartão de cidadão, existe a seguinte árvore de certificados digitais: Baltimore CyberTrust Root (CA raiz) → ECRaizEstado → Cartão do Cidadão → EC de Assinatura Digital Qualificada do Cartão do Cidadão. Contudo, se o certificado for efetivamente válido, o documento se não tiver sido alterado ou invalidado de alguma forma, é considerado íntegro, havendo assim confiança na sua utilização.

5.1.3 Assinaturas de documentos MsOffice

Com documentos em formatos DOC ou DOCX, mais precisamente, documentos *MSWord* começámos por tentar explorar a *Component Object Model* (COM) do *MSWord* e tentar controlar o documento a partir do programa, evitando assim que seja o utilizador a fazer a assinatura passo-a-passo na *interface* gráfica do *MSWord*, uma vez que este já permite a assinatura digital do documento, desde que sejam seguidos um conjunto de passos.

É desejável que uma vez assinados os documentos em *MSWord*, quando convertidos para PDF possam manter o documento assinado digitalmente. Mas tal não é possível, porque uma

assinatura digital contém uma *hash* dos *bytes* do documento, e os *bytes* do *MSWord* são muito diferentes dos *bytes* de documentos como os PDF. Mesmo mostrando o mesmo conteúdo, torna-se impossível manter a assinatura válida no documento PDF gerado. Ainda assim, uma possível ideia da plataforma é permitir gerar PDF a partir de um documento *MSWord*, controlando uma vez mais a respetiva COM. Após gerar o documento, efetua-se uma assinatura digital, nesse mesmo documento, utilizando a mesma chave privada que se utilizou para assinar o documento DOC/DOCX. Esta é uma forma de manter as conversões de diferentes formatos sempre assinadas, de forma a que se houver alguma alteração, esta seja detetada. Porém as assinaturas digitais do documento DOC/DOCX vão ser sempre diferentes das do PDF gerado a partir dele.

Voltando à API da "SecureBlackBox", esta também permite a assinatura em documentos do *Office* do *MSWindows*, logo foi também elaborada a assinatura digital e respetiva validação recorrendo a estes métodos. A formulação de uma assinatura implica, após inicializações próprias da API, carregar o documento para uma classe conveniente. Esta classe tem informação relevante sobre o documento, tal como, se este está ou não assinado e também sobre o tipo de documento que pode ser um *OpenXMLDocument*, um *OpenXPSDocument* ou um documento do tipo binário. Dependendo deste tipo de documento, tem de se definir um *handler* de assinatura apropriado que consiga pegar no documento *Office* e assiná-lo digitalmente. Para efetuar esta assinatura é necessário um certificado digital que permita aceder à sua chave privada através de uma senha. Este *handler* permite também adicionar um *timestamp* à assinatura para que lhe fique associado uma hora e um dia. Após isto basta fazer *flush* ao documento para que ele grave a assinatura digital consigo.

Para validar a assinatura nesta API são necessárias duas verificações: a da assinatura em si e a do certificado digital utilizado para assinar. Após carregar o documento e verificar, efetivamente, se este está ou não assinado, é necessário carregar o *handler* utilizado para assinar e determinar o seu tipo (*OpenXML, OpenXPS*...). Depois de saber o tipo de *handler* é carregado um *ValidationStatus* que permite verificar se a assinatura está ou não válida. A partir do *handler* é possível também tirar o certificado digital que foi utilizado para assinar o documento e também o *timestamp*. Com essa informação o certificado é validado para dar a informação ao utilizador se o certificado que este está a utilizar foi ou não válido na altura em que foi utilizado para assinar. O certificado pode ser considerado inválido por já ter passado a data de validade na data em que foi usado para assinar ou então por não estar assinado por uma CA considerada de confiança, ou então novamente por não ter uma cadeia ou uma árvore de CAs associada.

Qualquer documento pode ter mais do que uma assinatura digital e é depois dada a possibilidade de validar cada uma delas, porém neste caso estamos apenas a utilizar uma assinatura que corresponde ao certificado digital e à chave privada do *Smart Card*.

Neste caso, optámos novamente por utilizar os JARs e as DLLs da biblioteca "SecureBlackBox", pois estes fazem a assinatura digital sem ter que abrir visivelmente o documento. Controlando a COM do *Word* vemos o documento a abrir, pois no fim o utilizador tem de carregar num botão de confirmação da assinatura. Não dá para carregar nesse botão

programaticamente. O objetivo é assinar logo e guardar o documento sem que seja necessária qualquer interação com o utilizador.

5.1.4 Servidor

Como já fora referido na arquitetura o servidor foi desenvolvido na *framework* do .NET 4.5 e é uma aplicação *Web* desenvolvida em ASP.NET *Model-View-Controller* (MVC) 4.0 no IDE "Visual Studio 2012".

Para receber os ficheiros previamente assinados foi criado um serviço *Representational State Transfer* (REST) que permitisse receber ficheiros, apresentando um serviço HTTP POST e permitir o *download* do mesmo, apresentando um serviço HTTP GET. Esta arquitetura foi escolhida, pelo facto de não serem necessárias abstrações adicionais de protocolos baseados em troca de mensagens tais como o protocolo de serviços *Web* como o *Simple Object Access Protocol* (SOAP). Da forma como está criado, cada vez que chega um pedido de, por exemplo, um HTTP POST, o servidor abre uma nova *thread* para tornar todo o processo mais eficiente. Desta forma, não é necessário esperar que um ficheiro acabe de ser enviado para ser enviado um novo. Com a criação de várias *threads* podem-se enviar diversos ficheiros em simultâneo.

Para um acesso ao servidor a partir de uma máquina fixa, a aplicação encarrega-se de o Java *applet* no HTML, ou seja, do lado do cliente. No HTML só é apenas referenciada uma *tag* que contém a classe principal que gera o *applet* e é também referenciada a biblioteca Java, com o arquivo que contém todos os métodos e funcionalidades de assinatura. Para este arquivo ser construído, foi necessário construirmos um ficheiro JAR, que junta todos os componentes e bibliotecas externas do projeto num só ficheiro. É, portanto, mais seguro do que utilizar *javascript* puro como vimos no capítulo anterior.

Uma grande generalidade das aplicações móveis mais conhecidas e utilizadas, são invocadas pelo navegador *Web*, quando o utilizador está prestes a entrar num *WebSite* que tem uma aplicação móvel associada e instalada no dispositivo. Esta associação é efetuada através de um URL *schema*.

Desta forma, quando um URL é chamado, o dispositivo móvel verifica se há alguma aplicação móvel com uma declaração associada ao URL e pergunta ao utilizador se quer arrancar com ela. A ideia da utilização deste tipo de esquema baseou-se no facto de se querer que o utilizador esteja sempre a utilizar uma aplicação *Web*, mantendo sempre o maior número de funcionalidades na mesma. Desta forma, a seleção do ficheiro a ser assinado digitalmente é feita na mesma na *interface* da *Web* e é disponibilizado um URL com a informação do ficheiro selecionado. No caso de o acesso ser a partir de uma plataforma com um sistema operativo móvel utilizou-se este tipo de URL. Este esquema é o nível mais alto da estrutura de nomes URL, isto é, nos URLs, por exemplo, no URL "*http://teste.com*", o esquema é o "*http://*". Com estes esquemas podemos detetar aplicações móveis, lançar ou retomá-las, enviar-lhes comandos e dados, receber dados e *callbacks*. Por exemplo, se o dispositivo móvel chamar um URL

"*glintt://assinaturadigital.com*", a aplicação móvel pode ser arrancada, desde que tenha declarado no respetivo ficheiro de *manifest* uma etiqueta XML declarando o esquema, no caso, "*glintt*" e o host, que neste exemplo, seria "*assinaturadigital.com*". Este caso serve tanto para *Android* como para *iOS*.

Esse URL é enviado como parâmetro do esquema de modo a que a aplicação móvel, uma vez iniciada, saiba qual é o ficheiro e o selecione, assinando-o digitalmente e, por fim, o envie para o servidor principal. De seguida, a aplicação móvel termina e volta-se ao navegador *Web*. Continua a ser uma aplicação *Web*, mas que recorre a uma aplicação móvel para efetuar assinaturas digitais.

Em termos de desenvolvimento, optámos então por criar uma aplicação móvel para cada um dos sistemas móveis mais utilizados.

A primeira tentativa foi tentar recorrer à ferramenta "Titanium" que permite desenvolver aplicações nativas para *iOS*, *Android*, *BlackBerry* e *Windows*, baseando-se apenas num só código. Este código é desenvolvido na linguagem *javascript*. Como vimos anteriormente, para além de não haver bibliotecas de assinaturas digitais de ficheiros nesta linguagem, esta é extremamente insegura. Neste caso, para além de não haver bibliotecas de assinaturas em *javascript*, não estávamos a reaproveitar código já desenvolvido, o que é um dos objetivos, de forma a que todo o sistema se baseie no mesmo código, sendo o mais universal possível. Para além de otimizar o tempo de desenvolvimento, permite evitar erros de conversão das *hashes* dos *bytes* dos ficheiros, uma vez que cada linguagem os trata e converte de forma diferente. No caso de uma má conversão, as verificações das assinaturas poderiam não funcionar, devido a essas possíveis más conversões. Desta forma a seguinte tentativa foi tentarmos utilizar a ferramenta "Xamarin" que permite criar aplicações nativas em *iOS*, *Android*, *MacOS* e *Windows* desenvolvendo código C#. Esta abordagem já vai de encontro ao que queremos, porque o servidor está todo desenvolvido em C# e para testar as verificações das assinaturas, tivemos de se desenvolver os métodos de assinaturas digitais. Logo, seria apenas reutilizar o código. Acontece que esta ferramenta não é gratuita a partir de um certo tamanho (em *bytes*) do projeto, ou seja, se os tamanhos dos ficheiros de todo o projeto excederem um limite, a utilização da ferramenta passa a ser paga. No caso, quer-se um IDE gratuito. De seguida, tentámos explorar a ferramenta "JDeveloper", desenvolvida pela *Oracle*, onde é possível desenvolver-se aplicações para sistemas *iOS* e *AndroidOS* na linguagem de programação Java. Esta ferramenta é *open-source* e suporta os algoritmos e as bibliotecas já utilizadas nos outros sistemas operativos para efetuar as assinaturas digitais. Novamente, houve uma limitação, pois as bibliotecas de assinatura já desenvolvidas estão compiladas em Java 1.6.0, e a edição de Java do "JDeveloper" que permite o desenvolvimento das aplicações para *iOS* e *Android* apenas permite no máximo a utilização de Java 1.4.0.

No seguimento da investigação, encontrámos um IDE de características semelhantes, ou seja, *open-source* e que permitisse a partir de Java gerar uma aplicação para a plataforma *iOS*. Então, no caso do *Android*, reaproveitámos o código Java, pois *Android* é desenvolvido nessa

mesma linguagem. Foi apenas necessário mudar as bibliotecas de assinatura (JARs) apropriadas para o sistema operativo em causa. E nesta aplicação para desenvolvimento *iOS*, que tem por nome "RoboVM" conseguimos, efetivamente, desenvolver aplicações para *iOS* verdadeiramente nativas, utilizando a *User Interface* nativa do sistema operativo e tendo um acesso completo ao *hardware*. No caso esta ferramenta traduz *bytecode* de Java em nativo código "ARM" ou "x86". Não é utilizado nenhum interpretador nem nenhuma máquina virtual. Está também pronto a lançar as aplicações para a *App Store*, logo a aplicação final pode ser colocada na loja de aplicações apropriada. Basicamente, o "RoboVM" tem uma ponte entre Java e *Objective-C*, que torna possível que se utilizem os objetos e as classes de programação de *Objective-C* como qualquer outro objeto ou classe de Java. Foi esta a ferramenta utilizada e escolhida para desenvolver a componente para *iOS*. Para além de ser gratuita, também suporta as bibliotecas de Java na versão 1.6.0. Esta ferramenta foi integrada no IDE "Eclipse Kepler" e permite a criação de aplicações para *iOS* como um projeto "RoboVM". Aqui novamente uniformizámos o código Java utilizado na parte *Android* e na parte do Java *applet*. O código é exatamente o mesmo, o que permite centralizar as coisas, no sentido de se houver um erro detetado, é mais facilmente corrigido em todos os componentes. Tudo o que envolve o processo de assinaturas digitais em termos de código Java, é exatamente o mesmo para as diversas plataformas.

5.2 Smart Card

O *Smart Card* escolhido para testar o conceito previamente levantado foi o cartão do cidadão português. Este cartão tem dois pares de chaves criptográficas público-privadas, um apropriado apenas para autenticação, e outro par, apenas para ser utilizado em assinaturas digitais.

Para a utilização deste *Smart Card* foi necessária a instalação do *middleware* do cartão do cidadão que sirva de intermediário entre a aplicação e o cartão. Este *middleware* existe para uma arquitetura do computador de 32 e também de 64 bits. No caso, foi utilizado como teste o de 64, devido à arquitetura do computador. Após esta primeira instalação é necessária adicionar uma biblioteca JAR chamada "pteidlib" para que se possa aceder à API do cartão, utilizando assim os métodos e as funções necessárias de forma a obter a informação necessária. Nesta biblioteca os produtores do cartão do cidadão disponibilizam informação pública que qualquer entidade pode aceder como os dados pessoais (nome, altura, número de identificação, etc).

O fluxo de informação no *middleware* para *Windows*, por exemplo, é: Cartão do Cidadão → Leitor de Cartões → *Kernel* → *Windows System32* API → .NET → aplicação .NET. Com a instalação do *middleware*, ficam instalados componentes e bibliotecas na API do *Windows System32* ou no equivalente para os sistemas operativos com base em *Linux* ou *MacOS*, que são

utilizadas pela aplicação final como intermediário para aceder à informação do cartão. Estão então disponíveis *middlewares* para sistemas *Desktop*.

Ao inserirmos o *Smart Card* numa máquina *MSWindows*, *Linux* ou *Mac OS*, são instalados os certificados digitais na *Certificate Store* do sistema operativo em causa. Uma vez selecionado o certificado digital de assinatura digital, quando se assina o documento, o cartão aciona uma janela de segurança onde é pedido um PIN para aceder à chave privada associada a esse certificado digital. Quando o *Smart Card* é removido os certificados são retirados da *Store*.

A linguagem de programação Java não é uma linguagem desenvolvida pela *Microsoft*, logo um acesso à *Certificate Store* para obter o certificado do cartão não é facilmente efetuado como na *framework* do .NET. Logo, era necessário encontrar uma ponte de ligação entre o *applet* de Java e esta *Store*. É comum utilizar-se a *Java Native Interface* (JNI) para se fazer uma ponte entre esta linguagem e o sistema operativo *Windows*. A partir desta *interface* é possível carregar DLLs para a memória e ler os seus métodos e funcionalidades na linguagem Java. Acontece que esta DLL tem de estar já na máquina do cliente e o código teria de se adaptar a essa máquina. Porém, pedir ao utilizador para efetuar o *download* de um DLL, por muito que este confie no sistema é bastante arriscado, e, aliás, a maior parte dos antivírus iria bloquear esse *download*. Mesmo que não bloqueassem, seria um princípio de segurança quebrado. À partida, qualquer utilizador comum de sistemas informático sabe que descarregar um DLL pode ser uma ameaça, pois a maior parte dos vírus, vermes, cavalos de Tróia ou *rootkits* vêm neste formato. A solução passa por evitar que o *applet* dependa de algo de máquina para máquina. O objetivo é ser o mais universal possível e queremos que o sistema multiplataforma funcione em qualquer máquina, independente de certas localizações específicas de ficheiros ou de *downloads*. Logo, a solução foi tentar utilizar outra abordagem que não envolvesse a utilização desta DLL. O *middleware* do cartão do cidadão instala algumas bibliotecas internamente no sistema operativo e uma dessas bibliotecas lida com a parte da *Public-Key Cryptography Standards* (PKCS) do cartão. O PKCS é um grupo de normas concebido e publicado pela empresa "RSA Security Inc". Basicamente estes padrões promovem o uso de técnicas criptográficas tais como o algoritmo de cifra RSA e fazem a ligação à parte criptográfica no interior do cartão. No caso do nosso *Smart Card* de teste, o cartão do cidadão utiliza a PKCS #11. Esta versão é facilmente tratada na versão 1.8.0 da linguagem Java, pois esta é a versão mais recente e alberga as versões mais recentes de PKCS também. Então, acedeu-se a esse ficheiro que trata da PKCS do *Smart Card* e carregando os *bytes* para uma classe que trata desta informação no Java, ao assinar, no momento em que é necessária a utilização da chave privada, aparece o pedido de inserção do PIN que dá acesso a essa mesma chave. O *middleware* do cartão do cidadão instala este ficheiro que trata da PKCS nos sistemas operativos *Windows*, *Linux* e *MacOS*. A única diferenciação que teve de existir no código foi uma pequena bifurcação, dependendo do sistema operativo, onde é dado o caminho completo deste ficheiro. Por omissão, o *middleware* instala sempre no mesmo sítio, logo o caminho do ficheiro para todas as máquinas de cada sistema operativo é igual. Relembramos que este ficheiro é realmente necessário para se poder aceder à chave privada do *Smart Card*.

Enquanto que a *framework* .NET acede facilmente à *Certificate Store* do *Windows*, utilizando Java é necessário como auxílio recorrer a este ficheiro que acaba por aceder diretamente ao cartão, em vez de aceder à *Certificate Store* do sistema operativo. Desta feita o código-fonte é igual para todos os sistemas operativos fixos. Para acedermos a esse mesmo cartão é que é necessário o ficheiro que trata da PKCS, para fornecer os protocolos e a arquitetura interna do cartão. Esta informação está muito bem protegida, logo só através do acesso a este ficheiro é que, internamente, se consegue efetuar o acesso ao cartão.

Do lado das aplicações móveis, como vimos no capítulo da arquitetura optámos por executar uma chamada URL *schema* para aceder à aplicação móvel instalada nativamente. Porém, como não há um *middleware* desenvolvido pelos produtores do cartão do cidadão para ambientes móveis, a solução encontrada passou por se ter de inserir um código que diz respeito a uma máquina *Windows* que tem um leitor de cartões lá inserido. Esta solução passa a ser adotada no caso de outros *Smart Cards* que não o cartão do cidadão. Isto porque nesta abordagem o cartão do cidadão pede sempre o PIN no lado do servidor. Nesse caso, em caso de ataque ao servidor, poderiam obter o PIN do *Smart Card* de um médico e isso é algo que se quer evitar. Na solução da aplicação móvel nativa assinar, aí não há problema porque ao inserir o PIN este fica sempre no lado do cliente e não é do conhecimento da rede, mas no outro caso tal é indesejável, caso o servidor esteja comprometido.

5.3 Resumo

Concluindo, tendo em conta as restrições tecnológicas e de segurança, foi possível implementarmos um protótipo com o sistema a funcionar para todas as plataformas com o que foi proposto no início. Provámos que é possível haver segurança e se pode efetuar assinaturas digitais com um *Smart Card* num sistema único e centralizado, que contém métodos de criptografia, recorrendo às bibliotecas estudadas, que permitem efetuar as assinaturas digitais e as respetivas validações. As tecnologias de implementação foram o IDE "Eclipse Kepler" para a implementação do Java *applet* e o "Visual Studio 2012" para a parte do servidor central ASP.NET MVC (C#). O sistema implementado é uma aplicação *Web* que fornece ao utilizador *interfaces* adequadas ao sistema operativo em causa. Caso seja um cliente a entrar na aplicação com um sistema operativo fixo, ele fornece a *interface* do Java *applet*, caso contrário a *interface* é HTML desenvolvido nas vistas / *views* do ASP.NET.

De salientar ainda, que a partir do momento em que haja um desenvolvimento do *middleware* do cartão do cidadão para sistemas operativos móveis, a aplicação móvel retomará à opção de fazer um *callback* para a aplicação móvel nativa. A aplicação nativa acede ao cartão, assina o documento e envia-o para o servidor, tal como o Java *applet*. O utilizador não interage com a aplicação móvel, esperando apenas que a assinatura seja efetuada, garantindo assim que toda a interação pessoa-computador é única e exclusivamente com a aplicação *Web*.

5. Implementação

Tal como está desenvolvido, a aplicação *Web* adapta-se a qualquer *Smart Card*. As únicas alterações a fazer é definir o nome do certificado digital próprio para a assinatura e definir o caminho do ficheiro que trata do PKCS para os sistemas *Desktop*. Não é necessário mudar a infraestrutura a nível arquitetural, tanto a baixo como a alto nível. Basta apenas mudar o nome do certificado. Todos os *Smart Cards* ou instalam um *middleware*, que contêm os tais ficheiros de PKCS, ou então registam os seus certificados na *Certificate Store* do sistema e são facilmente acessíveis pelo C# na componente do servidor. Neste último caso só necessita mesmo de saber o nome do certificado digital próprio para efetuar a assinatura.

Capítulo 6

6. Validação dos Resultados

Neste capítulo será apresentada a maneira como se validam os sistemas desenvolvidos.

A validação foi feita passo por passo, isto é, começamos por testar pequenas funcionalidades em separado. Depois foram efetuados testes de carga elevada para garantir também que o sistema trata devidamente os diversos pedidos em simultâneo. Na medida em que se esperavam milhares de pedidos de assinaturas e envios de ficheiros para o servidor, num certo intervalo de tempo e, possivelmente em simultâneo, foi importante avaliarmos se o Sistema responde dentro de um tempo "razoável".

6.1 Testes a Casos de Utilização

De início, os casos de utilização mais testados foram os casos bases que garantem a segurança e a confiança da aplicação.

- O primeiro caso foi assinarmos um documento através de uma chave privada associada a um certificado digital, que é o caso de utilização mais básico. O teste foi efetuado nos mais diversos sistemas operativos: *MSWindows*, sistemas *Linux*, *MacOS*, *Android* e *iOS*. Em todos eles as assinaturas digitais foram efetuadas com sucesso. O primeiro caso de teste foi um ficheiro XML normal sem esquemas nem *namespaces* específicos e o método de assinatura foi o SHA de 512 bits com um certificado digital da empresa Glintt e com uma chave privada RSA, de 2048 bits, associada. Em todos os sistemas operativos a assinatura foi testada e foi efetuada com sucesso.

Foram efetuados os testes de assinatura em todas elas, porém, por uma questão de uniformização de todo o sistema e, consequentemente, do código-fonte, optamos pela implementação do "SecureBlackBox" tal como foi referido no capítulo da Implementação. Esta garante assinaturas e respetivas verificações em todos os tipos de documentos que pretendemos, permite o desenvolvimento para o *applet* de Java e para o lado do C# no servidor e é ainda consideravelmente eficiente e permite integração com o *Smart Card*, que é fundamental. Apesar do custo das licenças ser elevado, a Glintt está disposta a adquiri-la. Logo, por uma questão de um funcionamento eficiente e eficaz, optámos por esta biblioteca, como já tínhamos referido.

- De seguida, foram efetuados testes com o *Smart Card* já com a biblioteca "SecureBlackBox". Todos os sistemas operativos utilizaram o mesmo *Smart Card*, no caso o cartão de cidadão português, utilizando o seu certificado digital de assinatura digital, tendo que inserir um PIN cada vez que utilizavam a chave privada para efetuar a assinatura. Todas as assinaturas foram efetuadas como era pretendido. Para testarmos as componentes *Linux*, *MacOS*, *Android* e *iOS* nos simuladores de um computador com o sistema operativo *MacOS*, acedeu-se remotamente ao computador (*Windows*) que está a correr a componente de servidor, pois este disponibilizou acesso pelo *Internet Information Services* (IIS) *Express*. Este serviço permite que uma máquina normal possa funcionar de servidor *Web* possa ser acedido remotamente para efeitos de teste. Para isto foi também necessário criarmos uma regra de acesso na *firewall* que deixasse aceder pedidos a uma porta em específico. O *Website* está a correr em *localhost* e com uma determinada porta de acesso. É essa porta que devemos desbloquear na *firewall* para conseguirmos que os pedidos remotos, na mesma *intranet*, funcionem.

Antes de tentarmos validar um novo teste repetiam-se a validação dos anteriores, de modo a garantir que tudo continuava funcional.

- Um teste também básico foi abrirmos um documento previamente assinado e fazer a verificação se o documento foi ou não alterado. Se tiver mesmo sido alterado, o sistema deve detetar que o documento é inválido; se não tiver sido alterado, o sistema deve informar o utilizador que o documento está válido e apto a ser utilizado medicamente.

Foram efetuados os testes em ambiente *Windows* e *MacOS* com o mesmo Java *applet*, com o navegador *Web* "*Mozilla Firefox*". As assinaturas digitais e as validações foram efetuadas com os mesmos ficheiros em formato XML, PDF e *MSWord*. Tanto num sistema operativo como noutro, as assinaturas funcionaram corretamente em todos os formatos. A estratégia da utilização de um Java *applet* funcionara para sistemas operativos *desktop*. Para os sistemas móveis foram utilizados os simuladores apropriados e alguns, acedendo via *intranet* à máquina *Windows* que estava a fazer de servidor. O sucesso foi igual, sendo que se o documento tivesse sido adulterado após a assinatura isso seria detetado, caso contrário o documento era validado.

Desta vez, para cada um dos sistemas operativos foram utilizados os principais navegadores *Web*. Foi também um dos objetivos definidos no início da dissertação que a partir de qualquer navegador se pudesse aceder à aplicação *Web*. Em *MSWindows* experimentaram-se os navegadores "Internet Explorer", "Mozilla Firefox", "Opera", "Safari" e "Google Chrome", para sistemas *Linux* foram utilizados os navegadores "Mozilla Firefox", "IceWeasel", "Chromium" e "Google Chrome", para *MacOS* foram utilizados os navegadores "Mozilla Firefox", "Opera", "Safari" e "Google Chrome", para *Android* o navegador pré-definido e o

"Google Chrome" e para *iOS* foi utilizado o "Safari". Em todos os casos, o sistema manteve todas as funcionalidades de assinatura digital e respetiva validação.

6.2 Testes de Carga

Uma vez testados os casos base foi necessário definir métricas de qualidade em termos de tempo de execução. Estas métricas dependem do *hardware* utilizado e também da velocidade da rede. Tendo em conta estes fatores definiu-se então um tempo máximo esperado de acordo com os parâmetros utilizados e testou-se se o tempo está dentro dos valores estipulados. Este teste é apenas no envio de um ficheiro para o servidor. A lentidão do servidor pode, porventura, ser devido ao excesso de ficheiros simultâneos a enviar ficheiros.

Devido ao possível elevado número de acessos que o Sistema vai ter simultaneamente (tendo em conta que irão estar a trabalhar diversos médicos em simultâneo), foi necessário efetuar testes de carga, para se avaliar o funcionamento do Sistema, em termos de disponibilidade. Se estes testes não passarem é preciso rever toda a arquitetura do sistema e toda a implementação.

Para cada caso de uso pusemos a correr quinze mil *threads* a enviar documentos ao Sistema e este respondeu adequadamente dentro dos valores das métricas. O valor (quinze mil) foi considerado um valor já considerável, que simula, significativamente, o número de pedidos em tempo real. É um valor que, segundo dados da empresa Glintt, representa em média o número de acessos máximos em simultâneo que poderão ocorrer no sistema.

Aqui ficam os resultados das medidas do desempenho do sistema, conforme as métricas de qualidade:

Tabela 2: Performance do sistema no envio de ficheiros para o servidor

Velocidade de Rede (*Download* / *Upload*) em Mbps	Acessos em simultâneo	Ação	Tempo medido / Tempo máximo aceitável em ms
8.46 / 14.32	1	Assinatura Digital + *Upload* do ficheiro	606 / 2000
8.46 / 14.32	1	Verificação + *Upload* do ficheiro	549 / 2000
8.46 / 14.32	1	Assinatura + Verificação + *Upload* do ficheiro	898 / 3000
8.46 / 14.32	10	*Upload* do Ficheiro	939 / 1500

Tabela 2: Performance do sistema no envio de ficheiros para o servidor

8.46 / 14.32	500	*Upload* do Ficheiro	1082 / 2000
8.46 / 14.32	15000	*Upload* do Ficheiro	2034 / 4000
2.23 / 6.11	500	*Upload* do Ficheiro	1380 / 3000
2.23 / 6.11	15000	*Upload* do Ficheiro	2862 / 5000
72.43 / 52.03	500	*Upload* do Ficheiro	757 / 2000
72.43 / 52.03	15000	*Upload* do Ficheiro	1053 / 3000

De notar que as variáveis que fazem a diferença são a velocidade da rede, em termos de *download* e *upload*. O número de acessos em simultâneo ao servidor foi testado em números diferentes para que se percebesse bem a diferença de desempenho. Os tempos recolhidos são sempre calculados a partir de uma só máquina. Mesmo quando estão a haver 15000 acessos, o tempo contabilizado é o de espera numa máquina e não a soma dos tempos. O que interessa contabilizar é sempre o máximo de tempo que um médico espera para enviar o ficheiro para o servidor. Para cada tempo foram efetuadas 30 amostras, sendo que o resultado final é uma média entre todas elas.

As métricas de qualidade definidas de acordo com dados e estatísticas comummente utilizadas pela empresa (Glintt 2014b) foram:

Tabela 3: Métricas de qualidade de desempenho do sistema

Intervalo em Mbps	Número de acessos	Máximo de tempo permitido em ms
1 - 8	1	1000
1 - 8	500	3000
1 – 8	15000	5000
8 – 20	1	1000
8 – 20	500	2000
8 – 20	15000	4000

Tabela 3: Métricas de qualidade de desempenho do sistema

50 – 80	1	1000
50 – 80	500	2000
50 – 80	15000	3000

Estes valores são considerados tempos razoáveis para um médico no seu contexto profissional efetuar assinaturas digitais e fazer a sua respetiva validação. Queremos que os tempos definidos não afetem o tempo de trabalho do médico, daí termos definido apenas alguns segundos de espera durante a assinatura digital. O *hardware* das máquinas de onde todos estes valores foram retirados tem um processador, uma memória RAM e uma *motherboard* e placa de rede semelhantes aos utilizados no contexto da maior parte dos hospitais em Portugal e no Brasil (Glintt 2014c).

6.3 Testes de Usabilidade

Os testes de usabilidade previstos deveriam ser efetuados no final por médicos (utilizadores finais), e em situações reais. Todas as operações que dizem respeito a um utilizador final devem ser efetuadas para verificar que este consegue fazê-las de forma intuitiva, sem ajuda de terceiros e de forma rápida. Contudo, nesta fase do desenvolvimento da plataforma, ainda em protótipo, mal se acede à aplicação existe logo a possibilidade de escolher um documento e, de seguida, assiná-lo. Ainda não é um sistema virado para o utilizador final. Foi apenas focada a parte tecnológica e a constante procura do sistema ser multiplataforma, não estando este ainda um produto acabado. Contudo, ficam aqui visíveis nas figuras 7, 8, 9 e 10 *printscreens* da aplicação tal e qual como está. Relembro que necessita ainda de ser integrada e redesenhada em termos gráficos antes de ser utilizada em ambientes médicos.

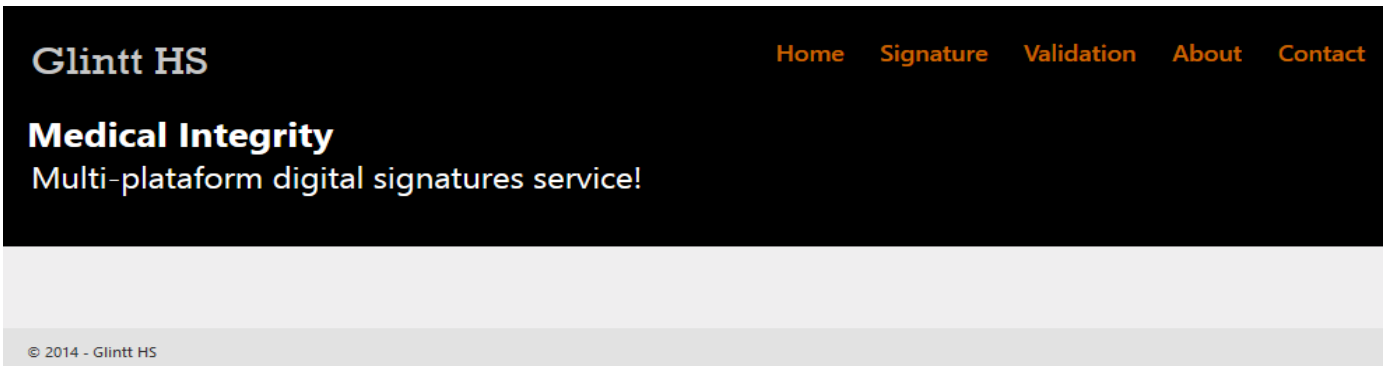

Figura 7: Página inicial da aplicação

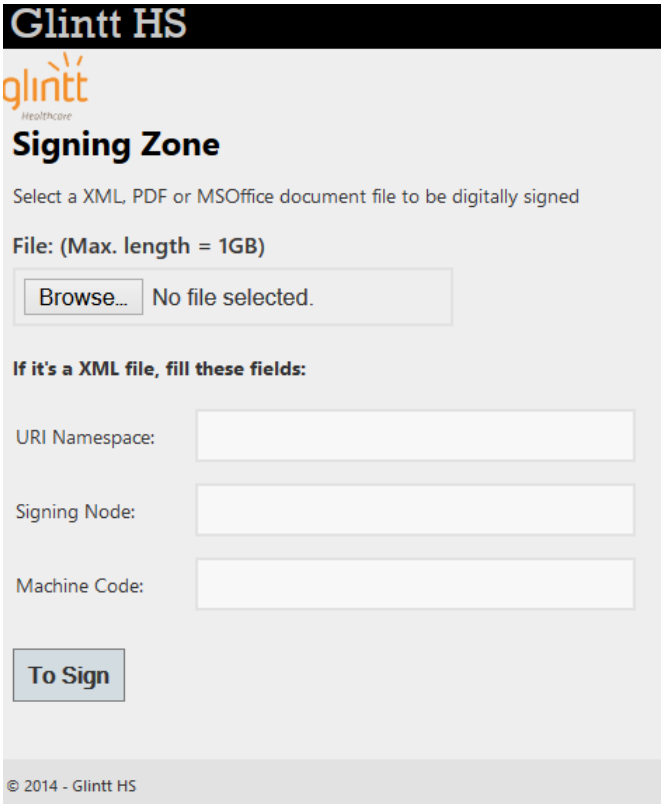

Figura 8: Vista de assinatura digital em dispositivos móveis

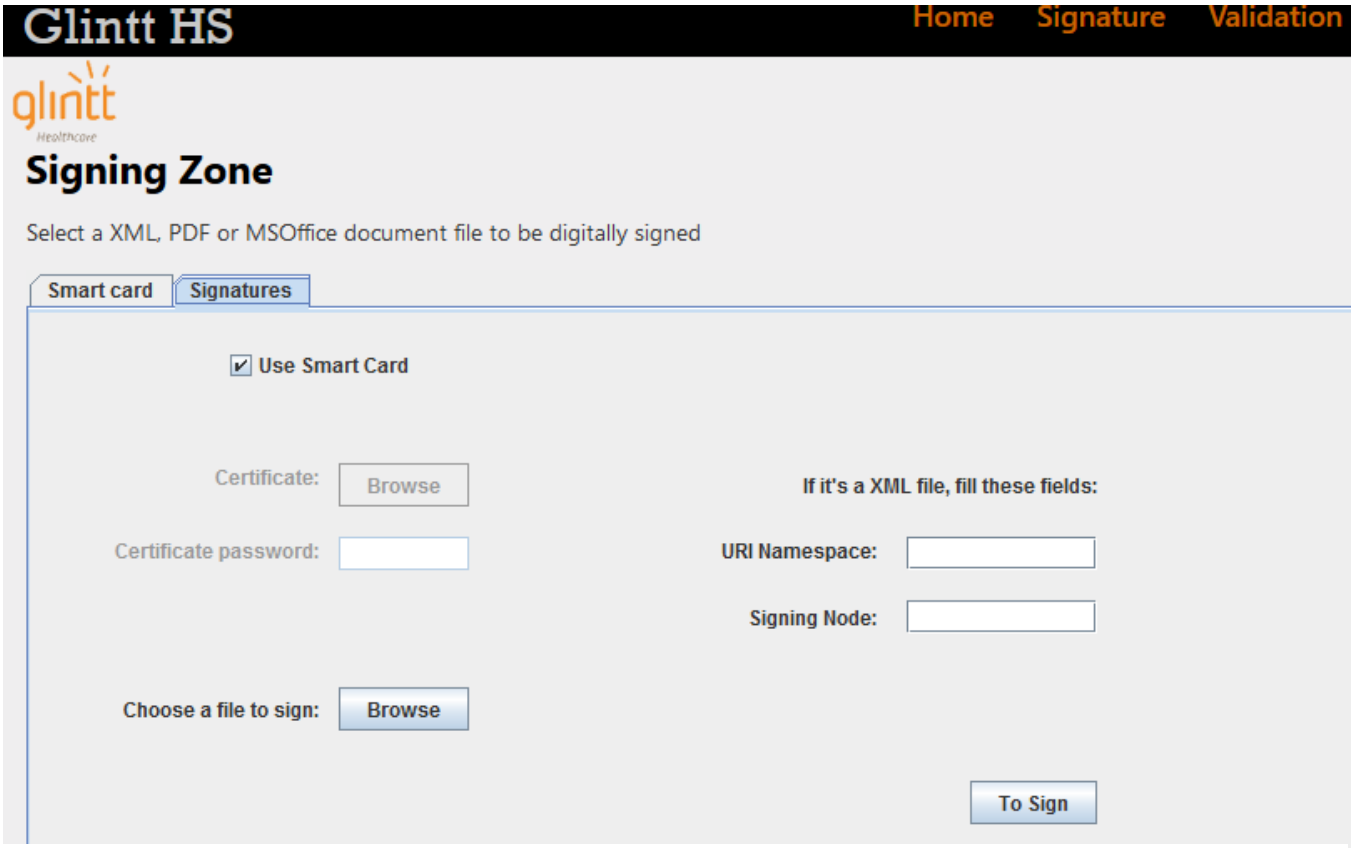

Figura 9: Java Applet para assinar digitalmente documentos

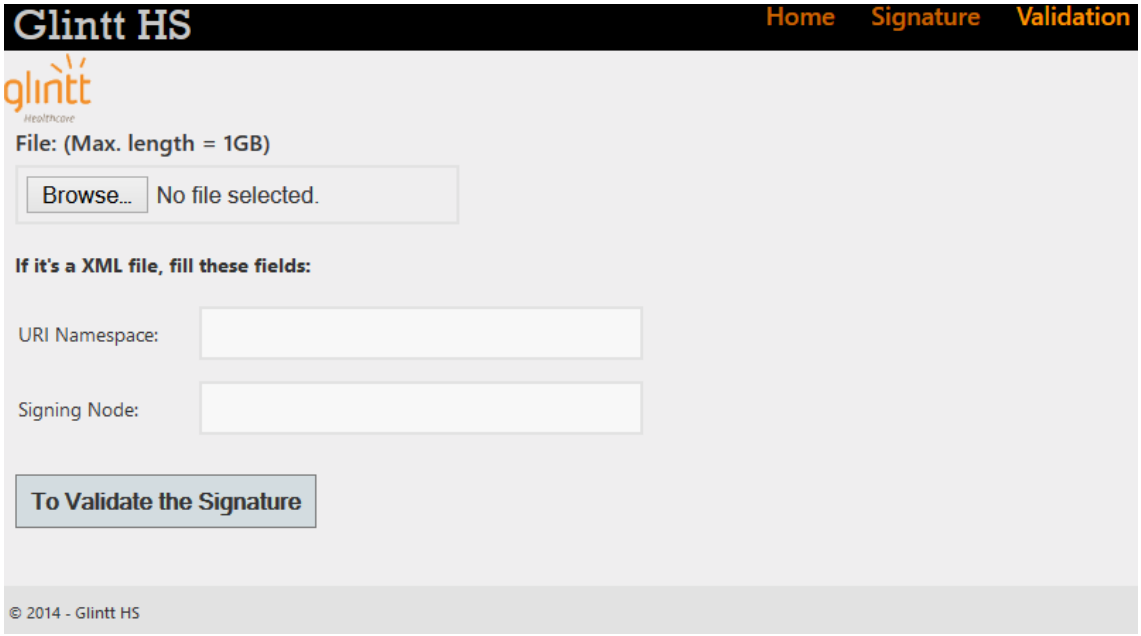

Figura 10: Validação da assinatura digital de um documento

Como podemos ver, a aplicação tem apenas as funcionalidades mais básicas.

Os testes de usabilidade principais por parte do médico são, primeiro, abrir um documento e alterá-lo e, de seguida, quando tiver que o assinar, inserir o *Smart Card* para que a assinatura seja efetuada e enviada ao sistema central. O médico deve perceber facilmente que tem de inserir o cartão para poder assinar digitalmente o documento em que está a trabalhar. De seguida, o teste de usabilidade mais importante é poder verificar se um documento, antes assinado, foi de alguma forma alterado; neste caso, o médico deve ser notificado de que aquele documento não é o original. Todo este processo tem de ser o mais simples e rápido possível.

6.4 Resumo

Para que haja uma correta validação dos resultados, que nos assegure com garantia que a hipótese inicial se verifica, é necessário dividir os testes em três tipos: os casos de utilização; os de carga; e os de usabilidade. Os testes aos casos de uso e aos de carga são aqueles que determinam realmente a correção do sistema, pois o principal foco da dissertação é que o sistema esteja funcional e preparado para um ambiente realista, reagindo de forma eficaz e eficiente aos pedidos que vão chegando. Os testes aos casos de uso passam por validar as pequenas funcionalidades mais específicas do sistema, de forma a garantirmos os requisitos funcionais. Os testes de carga são uma parte também muito importante na validação dos

resultados, pelo potencialmente número elevado de acessos em simultâneo ao sistema, numa situação real. Por último, os testes de usabilidade terão de ser efetuados, por parte dos utilizadores finais, isto é, os médicos, para que consigam realizar todas as operações de uma forma intuitiva e rápida.

Capítulo 7

7. Conclusões

O grande objetivo do trabalho de dissertação foi conseguirmos aliar um sistema que funcionasse para toda e qualquer plataforma / sistema sem violar princípios de segurança informática. A geração de assinaturas digitais envolve a utilização de chaves criptográficas privadas, ou seja, que dizem apenas respeito a uma só entidade e que só ela deve ter conhecimento. Por isso, teve de haver muito cuidado para garantir que essa informação não fosse acedida por entidades indesejadas. O facto de o sistema ser multiplataforma levantou algumas dificuldades em garantir o cumprimento de princípios de segurança.

Como vimos, por exemplo, no capítulo da arquitetura, ao princípio, a ideia era disponibilizar um serviço *Web* de assinatura digital, a partir do qual qualquer cliente com um navegador acedia e enviando um ficheiro, recebê-lo-ia, de seguida, assinado. Contudo, isso implicaria que o servidor *Web* tivesse conhecimento da chave privada do médico que deseja assinar o documento. A chave privada tem de estar sempre com a entidade associada e deve ser apenas utilizada localmente, não devendo ser enviada para a *Web*. De seguida, foi necessário então criar um componente de assinatura digital para o lado do cliente, para que a chave privada do médico fosse apenas utilizada localmente, e não saísse para a rede. Desenvolvemos então um Java *applet* para equipamentos fixos e aplicações nativas para sistemas móveis. Toda a parte de assinatura digital baseia-se no mesmo código-fonte, baseado na linguagem Java. Todo o código-fonte de assinaturas digitais foi centralizado e era comum a todos os sistemas. E, mais importante, aliada a toda esta segurança, conseguimos que a interação com a *interface* do sistema fosse apenas com a aplicação *Web*. No caso do Java *applet* é a aplicação *Web* que a demonstra, caso seja detetado que o sistema operativo do cliente seja do tipo *Desktop* e no caso das aplicações móveis, quando o médico escolhe o documento a assinar, é feita uma chamada interna à aplicação nativa, que a abre, utilizando-a para assinar o documento e enviá-lo ao servidor, terminando, de seguida.

7. Conclusões

Em suma, conseguimos desenhar e implementar um sistema que é multiplataforma, e funciona tanto nos sistemas operativos fixos como móveis. Para além disso, é um sistema acessível por qualquer navegador *Web*, sem restrições, o que dá ao médico um maior conforto na escolha da plataforma e do navegador.

Em termos de trabalho por realizar, é necessária toda a integração com uma CA que recorre a RAs e, possivelmente, a KRSs para que todos os certificados digitais dos médicos sejam devidamente autenticados e certificados pela CA, sendo que apenas os médicos com os certificados devidamente registados e válidos possam gerar assinaturas digitais válidas. Toda esta infraestrutura por trás é que torna o sistema numa PKI devidamente estruturada, equilibrada e eficiente. O foco da dissertação foi a geração e verificação de assinaturas digitais em ambientes multiplataforma.

Outra componente desejável, mas que está fora do âmbito do trabalho, seria permitir a utilização da plataforma por pacientes, isto é, permitir que um documento elaborado e certificado pelo médico seja, posteriormente, consultado pelo paciente. Um médico após elaborar, por exemplo, uma receita médica ou outro qualquer tipo de prescrição deve permitir que o documento seja visualizado pelo paciente em causa. No sistema implementado não foi desenvolvido o controlo de acessos com devidas permissões, por não estar previsto. Apenas o médico deve ter acesso a todos os seus documentos e o paciente deve ter acesso aos documentos que lhe dizem respeito. Essa componente de verdadeira utilização num contexto de uma clínica ou de hospital não foi desenvolvida, tendo só sido considerado um protótipo de assinatura e validação de documentos transacionados pelos médicos. Nesta componente, cada utilizador iria ter diferentes permissões de acesso, que poderiam ser detetados também através de certificados digitais de autenticação, e tendo em conta estas permissões, iriam ser estabelecidos diversos níveis de acesso nos quais se determinaria as ações que uma entidade tinha sobre determinado objeto do sistema.

Referências

[Chan 2000] Chan, Alvin T. S. 2000. "WWW+smart card: towards a mobile health care management system." *International Journal of Medical Informatics* no. 57 (2–3):127-137. doi: http://dx.doi.org/10.1016/S1386-5056(00)00061-7.

[Eldos 2014] Eldos. *SecureBlackBox* 2014. Available from https://www.eldos.com/sbb/.

[Fernandez-Aleman et al. 2013] Fernandez-Aleman, J. L., I. C. Senor, P. A. Lozoya, and A. Toval. 2013. "Security and privacy in electronic health records: a systematic literature review." *J Biomed Inform* no. 46 (3):541-62. doi: 10.1016/j.jbi.2012.12.003.

[Glintt 2014a] Glintt. 2014a. Comunicação privada da empresa sobre os tipos de documentos médicos.

[Glintt 2014b] Glintt. 2014b. Comunicação privada sobre as Métricas de qualidade comummente utilizadas pela empresa.

[Glintt 2014c] Glintt. 2014c. Comunicação privada sobre o tipo de hardware nos hospitais clientes da empresa.

[Gupta and Matyas Jr 2000] Gupta, Sarbari, and Stephen M. Matyas Jr. 2000. "Public Key Infrastructure: Analysis of Existing and Needed Protocols and Object Formats for Key Recovery." *Computers & Security* no. 19 (1):56-68. doi: http://dx.doi.org/10.1016/S0167-4048(00)86364-6.

[Hernandez-Ardieta, Gonzalez-Tablas, and Alvarez 2008] Hernandez-Ardieta, Jorge L., Ana I. Gonzalez-Tablas, and Benjamin Ramos Alvarez. 2008. "An optimistic fair exchange protocol based on signature policies." *Computers & Security* no. 27 (7-8):309-322. doi: 10.1016/j.cose.2008.07.005.

[iText 2014] iText. *iText, Programmable PDF Software* 2014. Available from http://itextpdf.com/.

[Khan 1998] Khan, Liaquat. 1998. "Deploying public key infrastructures." *Information Security Technical Report* no. 3 (2):18-33. doi: http://dx.doi.org/10.1016/S0167-4048(98)80002-2.

[Kravitz 1993] Kravitz, D.W. 1993. Digital signature algorithm. Google Patents.

[Lindberg and Rydin 2001] Lindberg, Henrik, and Pontus Rydin. 2001. Model view controller. Google Patents.

[Matasano 2014] Matasano. 2014. Javascript Cryptography Considered Harmful. Available from http://matasano.com/articles/javascript-cryptography/

Referências

[Mcelroy and Turban 1998] Mcelroy, D., and E. Turban. 1998. "Using smart cards in electronic commerce." *Int. J. Inf. Manag.* no. 18 (1):61-72. doi: 10.1016/s0268-4012(97)00040-6.

[Palomino and Palomino 2014] Palomino, Víctor Manuel Villa, and Luis Miguel Villa Palomino. *XAdES .NET Project* 2014. Available from http://xadesnet.codeplex.com/.

[QREN 2007] QREN. *Projecto Cartão de Cidadão - Especificações - Leitor Base* 2007. Available from http://www.pofc.qren.pt/ResourcesUser/2013/Concursos_Avisos/AAC01_SAMA_6_CarateristicasTecnicas_Leiroresbase_Cartao_de_Cidadao_v1.0.pdf.

[Ray and Biswas 2011] Ray, S., and G. P. Biswas. 2011. Design of Mobile-PKI for using mobile phones in various applications. Paper read at Recent Trends in Information Systems (ReTIS), 2011 International Conference on, 21-23 Dec. 2011.

[Rivest, Shamir, and Adleman 1978] Rivest, R. L., A. Shamir, and L. Adleman. 1978. "A method for obtaining digital signatures and public-key cryptosystems." *Commun. ACM* no. 21 (2):120-126. doi: 10.1145/359340.359342.

[Ruoyu, Gail-Joon, and Hongxin 2012] Ruoyu, Wu, Ahn Gail-Joon, and Hu Hongxin. 2012. Secure sharing of electronic health records in clouds. Paper read at Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom), 2012 8th International Conference on, 14-17 Oct. 2012.

[Sertifitseerimiskeskus 2003] Sertifitseerimiskeskus. *The Estonian ID Card and Digital Signature Concept* 2003. Available from http://www.id.ee/public/The_Estonian_ID_Card_and_Digital_Signature_Concept.pdf.

[Verschuren 1998] Verschuren, Ton. 1998. Smart access: strong authentication on the Web. In *Proceedings of the TERENA networking conference '98 on Towards networking and services in the year 2001*. Dresden, Germany: Elsevier Science Publishers B. V.

[Zhang 2003] Zhang, Q. 2003. *Intrusion Resilient Digital Signature Key Security in Mobile/wireless Networks*: Auburn University.